Maurice Margenstern (Ed.)

LNCS 3354

Machines, Computations, and Universality

4th International Conference, MCU 2004
Saint Petersburg, Russia, September 2004
Revised Selected Papers

Springer

Maurice Margenstern (Ed.)

Machines, Computations, and Universality

4th International Conference, MCU 2004
Saint Petersburg, Russia, September 21-24, 2004
Revised Selected Papers

Springer

Volume Editor

Maurice Margenstern
University of Metz, LITA, EA 3097
Île du Saulcy, 57045 Metz, France
E-mail: margens@sciences.univ-metz.fr

Library of Congress Control Number: 2005921802

CR Subject Classification (1998): F.1, F.4, F.3, F.2

ISSN 0302-9743
ISBN 3-540-25261-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Boller Mediendesign
Printed on acid-free paper SPIN: 11404262 06/3142 5 4 3 2 1 0

Preface

In this volume, the reader will first find the invited talks given at the conference. Then, in a second part, he/she will find the contributions which were presented at the conference after selection. In both cases, papers are given in the alphabetic order of the authors.

MCU 2004 was the fourth edition of the conference in theoretical computer science, *Machines, Computations and Universality*, formerly, *Machines et calculs universels*. The first and the second editions, MCU 1995 and MCU 1998, were organized by Maurice Margenstern, respectively in Paris and in Metz (France). The third edition, MCU 2001, was the first one to be organized outside France and it was held in Chişinău (Moldova). Its co-organizers were Maurice Margenstern and Yurii Rogozhin. The proceedings of MCU 2001 were the first to appear in Lecture Notes in Computer Science, see LNCS 2055.

From its very beginning, the MCU conference has been an international scientific event. For the fourth edition, Saint Petersburg was chosen to hold the meeting. The success of the meeting confirmed that the choice was appropriate.

MCU 2004 also aimed at high scientific standards. We hope that this volume will convince the reader that this tradition of the previous conferences was also upheld by this one. Cellular automata and molecular computing are well represented in this volume. And this is the case for quantum computing, formal languages and the theory of automata too. MCU 2004 also did not fail its tradition to provide our community with important results on Turing machines. Also a new feature of the Saint Petersburg edition was the contributions on analog models and the presence of unconventional models.

Here is an opportunity for me to thank the referees of the submitted papers for their very efficient work. The members of the program committee gave me decisive help on this occasion. Thanks to them, namely Anatoly Beltiukov (co-chair), Erzsebet Csuhaj-Varjú, Nikolai Kossovskii (co-chair), Kenichi Morita, Gheorghe Păun, Yurii Rogozhin and Arto Salomaa, I can offer the reader this issue of LNCS.

The local organizing committee included Anatoly Beltiukov, Nikolai Kossovskii, Michail Gerasimov, Igor Soloviov, Sorin Stratulat and, especially, Elena Novikova.

MCU 2004 could not have been held without decisive supports. For this reason, I thank the *Laboratoire d'Informatique Théorique et Appliquée*, *LITA*, the University of Metz, and one of its faculties, UFR MIM.

Metz, 29 November 2004 — Maurice Margenstern

Organization

MCU 2004 was organized by the Laboratoire d'Informatique Théorique et Appliquée (LITA), University of Metz, Metz, France and the Euler International Mathematical Institute, part of the Saint Petersburg Department of the Steklov Institute of Mathematics, Russia.

Program Committee

Anatoly Beltiukov	**Co-chair**, Udmurt University, Izhevsk, Russia
Erzsebet Csuhaj-Varju	Hungarian Academy of Sciences, Hungary
Nikolai Kossovski	Saint Petersburg State University, Russia
Maurice Margenstern	**Co-chair**, LITA, University of Metz, France
Kenichi Morita	Hiroshima University, Japan
Yurii Rogozhin	Institute of Mathematics and Computer Science, Chişinău, Moldova
Arto Salomaa	Academy of Finland and Turku Centre for Computer Science, Finland

Sponsoring Institutions

Laboratoire d'Informatique Théorique et Appliquée (LITA), University of Metz, Metz, France and the UFR MIM.

Table of Contents

When a distinguished but elderly scientist states that something is possible, he is almost certainly right. When he states that something is impossible, he is almost certainly wrong. Arthur C. Clarke

Algorithmic Randomness, Quantum Physics, and Incompleteness

Cristian S. Calude

Department of Computer Science
University of Auckland, New Zealand
cristian@cs.auckland.ac.nz

Abstract. Is randomness in quantum mechanics "algorithmically random"? Is there any relation between Heisenberg's uncertainty relation and Gödel's incompleteness? Can quantum randomness be used to trespass the Turing's barrier? Can complexity shed more light on incompleteness? In this paper we use variants of "algorithmic complexity" to discuss the above questions.

1 Introduction

Whether a U_{238} nucleus will emit an alpha particle in a given interval of time is "random". If we collapse a wave function, what it ends of being is "random". Which slit the electron went through in the double slit experiment, again, is "random".

Is there any sense to say that "random" in the above sentences means "truly random"? When we flip a coin, whether it's heads or tails looks random, but it's not truly random. It's determined by the way we flip the coin, the force on the coin, the way force is applied, the weight of the coin, air currents acting on it, and many other factors. This means that if we calculated all these values, we would know if it was heads or tails without looking. Without knowing this information—and this is what happens in practice—the result *looks* as if it's random, but *it's not truly random.*

Is quantum randomness "truly random"? Our working model of "truly random" is "algorithmic randomness" in the sense of Algorithmic Information Theory (see, for example, [5]). In this paper we compare quantum randomness with algorithmic randomness in an attempt to obtain partial answers to the following questions: Is randomness in quantum mechanics "algorithmically random"? Is there any relation between Heisenberg's uncertainty relation and Gödel's incompleteness? Can quantum randomness be used to trespass the Turing's barrier? Can complexity cast more light on incompleteness? Our analysis is tentative and raises more questions than offers answers.

2 Algorithmic Randomness

The main idea of *Algorithmic Information Theory* (shortly, AIT) was traced back in time (see [15,16]) to Leibniz, 1686 ([36], 40–41). If we have a finite set of points (e.g. say, observations of an experiment), then one can find many mathematical formulae each of which produces a curve passing through them all, in the order that they were given. Can we say that the given set of points satisfy the "law" described by such a mathematical formula? If the set is very large and complex, and the formula is comparatively simpler, then, indeed, we have a law. In Chaitin's words [15], "a scientific theory is a computer program that calculates the observations, and that the smaller the program is, the better the theory."[1] If the mathematical formula *is not* substantially simpler than the data itself, then we don't have a law; still, there may be another mathematical formula qualifying as "law" for the given set. If *no* mathematical formula is substantially simpler than the set itself, the set is unstructured, law-less. Using the computer paradigm, if *no program is substantially simpler than the set itself, then the set is "algorithmically random"*.

Of course, to make ideas precise we need to define the basic notions, complex finite set, (substantially) smaller program, etc. A convenient way is to code all objects as binary strings and use Turing machines as a model of computation.

For technical reasons (see [5,23]), our model is a *self-delimiting Turing machine*, that is a Turing machine C which processes binary strings into binary strings and has a prefix-free domain: if $C(x)$ is defined and y is either a proper prefix or an extension of x, then $C(y)$ is not defined. The self-delimiting Turing machine U is *universal* if for every self-delimiting Turing machine C there exists a fixed binary string p (the simulator) such that for every input x, $U(px) = C(x)$: either both computations $U(px)$ and $C(x)$ stop and, in this case they produce the same output or both computations never stop. Universal self-delimiting Turing machines can be effectively constructed. The relation with computability theory is given by the following theorem:

> *A set is computably enumerable* (shortly, *c.e.*) iff *can be generated by some self-delimiting Turing machine.*

The Omega number introduced in [13]

$$\Omega_U = \sum_{U(x) \text{ stops}} 2^{-|x|} = 0.\omega_1\omega_2\ldots\omega_n\ldots \tag{1}$$

is the halting probability of U; $|x|$ denotes the length of the (binary) string x. Omega is one of the most important concepts in algorithmic information theory (see [5]).

[1] The modern approach, equating a mathematical formula with a computer program, would probably not surprise Leibniz, who designed a succession of mechanical calculators, wrote on the binary notation (in 1679) and proposed the famous "let us calculate" dictum; see more in Davis [20], chapter one.

The program-size complexity induced by C is defined by $H_C(x) = \min\{|w| : C(w) = x\}$ (with the convention that strings not produced by C have infinite complexity). The complexity H_C measures the power of C to compress strings. For example, if $H_C(x) \leq |x| - k$, then C can compress at least k bits of x; if $H_C(x) > |x| - k$, then C cannot compress more than $k - 1$ bits of x. A string x is algorithmically k–random with respect to C if the complexity $H_C(x)$ is maximal up to k among the complexities of all strings of the same length, that is, $H_C(x) \geq \max_{|y|=|x|} H_C(y) - k$.

One might suppose that the complexity of a string would vary greatly between choices of self-delimiting Turing machine. The complexity difference between C and C' is at most the length of the shortest program for C' that simulates C. Complexities induced by some self-delimiting Turing machines (called *universal*) are almost optimal, therefore, the complexity of a string is fixed to within an additive constant. This is the "invariance theorem" (see [5], p. 36):

For every self-delimiting universal Turing machine U and self-delimiting Turing machine C there exists a constant $\varepsilon > 0$ (which depends upon U and C) such that for every string x, $H_U(x) \leq \varepsilon + H_C(x)$.

In what follows we will fix a self-delimiting universal Turing machine U, write H instead of H_U, and use the term "machine" to denote a "self-delimiting Turing machine".

Algorithmic random strings are defined as above using U instead of C. This approach can be extended to "algorithmic random sequences" by requiring that the initial prefixes of the sequence cannot be compressed with more than a fixed number of bits, i.e. they are all "almost random": A sequence $\mathbf{x} = x_1x_2\ldots, x_n\ldots$ is *algorithmic random* if there exists a positive constant $c > 0$ such that for all $n > 0$, $H(x_1x_2\ldots, x_n) \geq n - c$. Chaitin's theorem [13] states that

The bits of Ω_U (i.e. the sequence $\omega_1\omega_2\ldots\omega_n\ldots$ in (1)*) form an algorithmic random sequence.*

3 From Algorithmic Randomness to Uncertainty

The randomness of quantum processes has been an integral part of the interpretation of quantum phenomena almost from the very outset of the theory. In fact, quantum physics is the only theory within the fabric of modern physics that integrates and is based on randomness. Quantum randomness has been confirmed over and over again by theoretical and experimental research conducted in physics since the first decades of the 20th century.[2]

But there is a problem: quantum randomness is postulated and is not at all a mathematical consequence of the standard model of quantum mechanics.

[2] The conclusion of [4] is : "We find no evidence for short- or long-term correlations in the intervals of the quantum jumps or in the decay of the quantum states, in agreement with quantum theory".

We don't know whether the randomness of quantum mechanics is genuine or simply an artefact of the particular mathematical apparatus physicists employ to describe quantum phenomena. Being pragmatic, perhaps we can accept the randomness because of the immense success of the applications of quantum mechanics. But even here there is room for doubt. As Wolfram ([56], p. 1064) has pointed out, "a priori, there may in the end be no clear way to tell whether randomness is coming from an underlying quantum process that is being measured, or from the actual process of measurement."

So, *what is the relation between algorithmic randomness and quantum randomness?* A detailed discussion appears in Svozil [53] (see also [52]). Yurtsever [57] argued that a string of quantum random bits is, *almost certainly*, algorithmically random. Here we take a different approach.

First and foremost, there is a strong *computational* similarity: both algorithmic and quantum randomness are *uncomputable*, they cannot be generated/simulated by any machine.[3] From this point of view, both types of randomness are fundamentally different from "deterministic chaos" (computable systems in which unobservably small causes can produce large effects) or pseudo-random numbers (generated by software functions; an elegant solution is the so-called "rule 30" discovered by Wolfram [55]).

The strong uncomputability of algorithmic randomness is expressed by the theorem:

The set of algorithmic random strings is immune.

That is, *no infinite set of algorithmic random strings is c.e.* (see [5], p. 119). In particular, the set of prefixes of a random infinite sequence is immune, hence the sequence itself is uncomputable.

Quantum randomness is postulated by Born's measurement postulate: *When a closed quantum physical system in state* $V = (v_{1,1}, v_{2,1}, \ldots, v_{n,1})^T$ *is* measured *it yields outcome i with probability* $|v_{i,1}|^2$. In this sense, according to Milburn (see [39], p. 1), the "physical reality is irreducibly random". For Peres [44], "in a strict sense quantum theory is a set of rules allowing the computation of probabilities for the outcomes of tests which follow specific preparations". In the standard model (Copenhagen interpretation) of quantum physics, quantum processes cannot be simulated on a classical Turing machine, not even on a probabilistic Turing machine (in which the available transitions are chosen randomly with equal probability at each step). The reason is Bell's Theorem, which, in Feynman's words ([26], p. 476), reads: *"It is impossible to represent the results of quantum mechanics with a classical universal device."*

A recently proposed complexity-theoretic analysis [11] of Heisenberg's uncertainty principle (see [29]) reveals more facts. The uncertainty principle states

[3] It is also very difficult for humans to produce random digits; based on 'history', computer programs can predict, on average, some of the digits humans will write down.

that the *more precisely the position is determined, the less precisely the momentum is known in this instant, and vice versa.* In its exact form (first published by Kennard [33]), for all normalized state vectors $|\Psi\rangle$,

$$\Delta_p \cdot \Delta_q \geq \hbar/2,$$

where Δ_p and Δ_q are standard deviations of momentum and position, i.e.

$$\Delta_p^2 = \langle\Psi|p^2|\Psi\rangle - \langle\Psi|p|\Psi\rangle^2;\ \Delta_q^2 = \langle\Psi|q^2|\Psi\rangle - \langle\Psi|q|\Psi\rangle^2.$$

For our analysis it is more convenient to define a variation of the program-size complexity, namely the complexity measure $\nabla_C(x) = \min\{N(w) \mid C(w) = x\}$, the smallest integer whose binary representation produces x via C. Clearly, for every string x,

$$2^{H_C(x)} \leq \nabla_C(x) \leq 2^{H_C(x)+1} - 1.$$

Therefore we can say that $\Delta_C(x)$, the uncertainty in the value $\nabla_C(x)$, is the difference between the upper and lower bounds given, namely $\Delta_C(x) = 2^{H_C(x)}$.

The invariance theorem can now be stated as follows:

For every universal machine U and machine C there exists a constant $\varepsilon > 0$ (which depends upon U and C) such that for every string x, $\Delta_U(x) \leq \varepsilon \cdot \Delta_C(x)$.

Let $\Delta_s = 2^{-s}$ be the probability of choosing a program of length s. Chaitin's theorem (cited at the end of Section 2) stating that the bits of Ω_U in (1) form a random sequence can now be presented as a "formal uncertainty principle":

For every machine C there is a constant $\varepsilon > 0$ (which depends upon U and C) such that

$$\Delta_s \cdot \Delta_C(\omega_1 \ldots \omega_s) \geq \varepsilon. \tag{2}$$

The inequality (2) is an uncertainty relation, as it reflects a limit to which we can simultaneously increase both the accuracy with which we can approximate Ω_U and the complexity of the initial sequence of bits we compute; it relates the uncertainty of the output to the size of the input. When s grows indefinitely, Δ_s tends to zero in contrast with $\Delta_C(\omega_1 \ldots \omega_s)$ which tends to infinity; in fact, the product is not only bounded from below, but increases indefinitely (see also [11]). From a complexity viewpoint (2) tells us that there is a limit ε up to which we can uniformly compress the initial prefixes of the binary expansion of Ω_U.

The above "formal uncertainty principle" (much like Heisenberg's uncertainty principle) is a general one; they both apply to *all* systems governed by the wave equation, not just quantum waves. We could, for example, use sound waves instead of a quantum system by playing two pure tones with frequencies f and $f + \Delta_C(\omega_1 \ldots \omega_s)$. Then Δ_s corresponds to the complementary observable, the length of time needed to perceive a beat.

For the remainder of this section *we assume that quantum randomness is algorithmic randomness.*[4]

The two conjugate coordinates are the random real and the binary numbers describing the programs that generate its prefixes. Then, the uncertainty in the random real given an n-bit prefix is 2^{-n}, and the uncertainty in the size of the shortest program that generates it is, to within a multiplicative constant, 2^n.

The Fourier transform is a lossless transformation, so all the information contained in the delta function $\delta_{\Omega(x)} = 1$ if $x = \Omega$, $\delta_{\Omega(x)} = 0$, otherwise, is preserved in the conjugate. Therefore, if you need n bits of information to describe a square wave convergent on the delta function, there must be n bits of information in the Fourier transform of the square wave. Since both the information in the transformed square wave and the shortest program describing the square wave increase linearly with n, there is an equivalence between the two.

Is (2) a 'true' uncertainty relation? We can prove that the variables Δ_s and Δ_C in (2) are standard deviations of two measurable observables in suitable probability spaces, see [11]. For Δ_s we consider the space of all real numbers in the unit interval which are approximated to exactly s digits. Consider the probability distribution $Prob(v) = P_C(v)/\Omega_C^s$, where $P_C(x) = \sum_{C(y)=x} 2^{-|y|}$ and $\Omega_C^s = \sum_{|x|=s} P_C(x)$. For Δ_C we consider

$$\beta = (\Delta_C(\omega_1\omega_2\ldots\omega_s))^{1/2} \cdot (Prob(\omega_1\omega_2\ldots\omega_s))^{-1/2} \cdot (1 - Prob(\omega_1\omega_2\ldots\omega_s))^{-1/2},$$

and the same space but the random variable $Y(\omega_1\omega_2\ldots\omega_s) = \beta$ and $Y(v) = 0$ if $v \neq \omega_1\omega_2\ldots\omega_s$. Hence, the relation (2) becomes:

$$\sigma_X \cdot \sigma_Y = \Delta_s \cdot \Delta_C(\omega_1\omega_2\ldots\omega_s) \geq \varepsilon.$$

For example, it is possible to construct a special universal machine $C = U_0$ satisfying the inequality $\Delta_s \cdot \Delta_{U_0}(\omega_1\ldots\omega_s) \geq 1$, for which we have:

$$\sigma_X \cdot \sigma_Y \geq 1.$$

The complexity-theoretic formulation of uncertainty has more "physical meaning", as shown in [11]. If the halting probability of the machine is computable, then we can construct a quantum algorithm to produce a set of qubits

[4] This is a disputable assumption. Bohm's interpretation says there are real particles with trajectories determined by a non-local equation, and the randomness is due to our ignorance about the state of the rest of the universe. Penrose says that the wave collapse is deterministic, but uncomputable and occurs when the difference between superposed space-times gets too large. Fredkin, following a tradition that goes back to Schrödinger and Einstein, says the wave collapse is computable and, probably, just a simple pseudo-random function; we have no idea what the structure of space is like at the Planck scale, which is only about 2^{-116} metres. Another view sees the classical world as emerging from the collisional interactions of quantum particles that inherently arise in "hot dense matter". Collisions destroy the purity of otherwise coherent states, so quantum randomness (as well as deterministic chaos) may be a manifestation of the incompleteness of dynamical laws, cf. [41].

whose state is described by the distribution. To illustrate, we consider a quantum algorithm with two parameters, C and s, where C is a machine for which the probability of producing each s-bit string is computable. We run the algorithm to compute that distribution on a quantum computer with s output qubits; it puts the output register into a superposition of spin states, where the probability of each state $|v\rangle$ is $P_C(v)/\Omega_C^s$. Next, we apply the Hamiltonian operator $H = \beta|\omega_1 \ldots \omega_s\rangle\langle\omega_1 \ldots \omega_s|$ to the prepared state. A measurement of energy will give β with probability $P = Prob(\omega_1\omega_2 \ldots \omega_s)$ and zero with probability $1 - P$. The expectation value for energy, therefore, is exactly the same as that of Y, but with units of energy, i.e.

$$\Delta_C(\omega_1\omega_2 \ldots \omega_s)[J] \cdot \Delta_s \geq \varepsilon[J],$$

where $[J]$ indicates Joules of energy.

Now define

$$\Delta_t \equiv \frac{\sigma_Q}{|d\langle Q\rangle/dt|},$$

where Q is any observable that does not commute with the Hamiltonian; that is, Δ_t is the time it takes for the expectation value of Q to change by one standard deviation. With this definition, the following is a form of Heisenberg's uncertainty principle:

$$\Delta_E \cdot \Delta_t \geq \hbar/2.$$

We can replace Δ_E by $\Delta_C(\omega_1\omega_2 \ldots \omega_s)$ by the analysis above; but what about Δ_t? If we choose a time scale such that our two uncertainty relations are equivalent for a single quantum system corresponding to a computer C and *one* value of s, then the relation holds for C and *any* value of s:

$$\Delta_C(\omega_1\omega_2 \ldots \omega_s)[J] \cdot \Delta_s \frac{\hbar}{2\varepsilon}[J^{-1} \cdot Js] \geq \frac{\hbar}{2}[Js].$$

In this sense, Heisenberg's uncertainty relation is equivalent to (2).

The uncertainty principle now says that getting one more bit of Ω_U requires (asymptotically) twice as much energy. Note, however, that we have made an arbitrary choice to identify energy with complexity. We could have chosen to create a system in which the position of a particle corresponded to the complexity, while momentum corresponded to the accuracy of C's estimate of Ω_U. In that case, the uncertainty in the position would double for each extra bit. Any observable can play either role, with a suitable choice of units.

If this were the only physical connection, one could argue that the result is merely an analogy and nothing more. However, consider the following: let ρ be the density matrix of a quantum state. Let R be a computable positive operator-valued measure, defined on a finite-dimensional quantum system, whose elements are each labelled by a finite binary string. Then the statistics of outcomes in the quantum measurement is described by R: $R(\omega_1 \ldots \omega_s)$ is the measurement outcome, and $tr(\rho R(\omega_1 \ldots \omega_s))$ is the probability of getting that outcome when we measure ρ. Under these hypotheses, Tadaki's inequality (1) (see [54], p. 2), and the relation (2) imply the existence of a constant τ (depending upon R) such that for all ρ and s we have:

$$\Delta_s \cdot \frac{1}{tr(\rho R(\omega_1 \ldots \omega_s))} \geq \tau.$$

In other words, there is no algorithm that, for all s, can produce an experimental set-up to produce a quantum state and a POVM (positive operator valued measure) with which to measure the state such that the probability of getting the result $\omega_1\omega_2 \ldots \omega_s$ is greater than $\tau/2^s$.

The above analysis is just one small step towards understanding the nature of quantum randomness—more theoretical and experimental facts are needed. One possible avenue of attack might be to experimentally test whether quantum random bits satisfy some properties proven for algorithmic random strings. For example, one such natural property states the existence of a constant c such that for every n, the number of algorithmically random strings of length n is greater than 2^{n-c}.

4 Randomness and Computation

As we have seen, there is no such thing as "software generated" genuine randomness. In John von Neumann's words: "Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin". On the other hand, randomness in quantum mechanics is hardly news. So what prevented quantum physics from becoming a dominant source of randomness?

Basically, *practical engineering considerations*. Until recently, the only quantum random number generators were based on the observation of radioactive decay in an element like radium. The first book containing a million of quantum random digits—generated by using radioactive decay from electronic vacuum tubes—was published by the RAND Corporation in 1955, [47]. The basic table was produced during May and June 1947; exhaustive tests found small but statistically significant biases and adjustments were made. Some of the early methods can be found in Golenko [28] who describes noise generators based on a germanium triode, on a gas-discharge tube with magnet, on an electronic trigger circuit with a switch in its anode supply (photograph in Fig. 44), on a gasotron with magnet, and on subharmonic generators. But such generators are quite bulky and the use of radioactive materials may cause health problems.

Fortunately, a beam of light offers an excellent alternative source of randomness (see [32]). Light consists of elementary particles called photons; they exhibit in certain situations a random behaviour. The transmission upon a semitransparent mirror is an example. A photon generated by a source beamed to a semitransparent mirror is reflected or transmitted with 50 per cent chance (see Fig. 1), and these measurements can be translated into a string of quantum random bits. Such a device can be (and was) manufactured, and its functioning can be monitored in order to confirm that is operating properly.

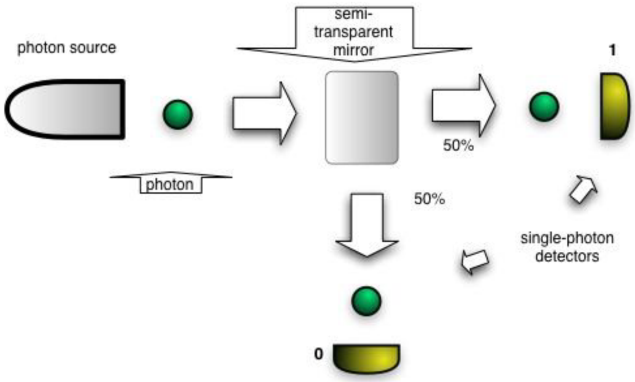


Fig. 1. Optical system for generating quantum random bits

A spin-off company from the University of Geneva, *id Quantique*, [30], markets a quantum mechanical random number generator called *Quantis*, see Fig. 2. *Quantis* is available as an OEM component which can be mounted on a plastic circuit board or as a PCI card; it can supply a (theoretically, arbitrarily) long string of quantum random bits sufficiently fast for cryptographic applications.[5]

Fig. 2. Quantis: OEM and PCI, cf. [31]

Plug these quantum random bits into a PC and we can, in theory at least, leapfrog Turing's barrier, that is *we obtain a computing device with capability surpassing that of classical Turing machines.* Indeed, as we have already noticed, no Turing machine can generate quantum random bits! So, the above statement is true *independently* of whether quantum random bits are or not algorithmically random.

[5] *id Quantique also supplies quantum random numbers over the internet [42], as well as HotBits, [27], which generates them via radioactive decay.*

Is this interesting? For some authors, the analysis of this type of ‘oracle’ machine is pointless and “one can only pity those engaged in this misguided enterprise” (cf. [21], p. 207). As the reader arriving at this point can expect, I do not share this view.

First, it seems that the computing device “PC plus a quantum generator of random bits”, whose existence can be hardly doubted, is a serious threat to the Church-Turing Thesis, which, in one variant, states that *every effective computation can be carried out by a Turing machine.*

Secondly, understanding this device may help coping with complex computations. Here is a relevant example. Testing whether a number is prime—showing that it has no factors beside itself and 1—is a crucial process in cryptography, and although theoretically fast deterministic algorithms for primality testing have been discovered (see [1][6]), in practice they are quite slow and do not pose any immediate risk to the security of electronic communication.

Probabilistic algorithms, first discovered in the mid 1970s, [43,46], can help speed things up, but such probabilistic tests—which essentially use a coin-flipping source of pseudo-random bits to search for a number’s factors—are only “probably” correct.[7] If you run the probabilistic algorithm using a source of algorithmically random bits, however, it would not only be fast, it would also *be correct every single time* (cf [17]). One of the principal tools used in computer simulation, known as fast Monte-Carlo algorithms, can derive a similar benefit from the use of algorithmically random numbers (cf. [12]; see more in [5]). *It is an open question whether these results are true for quantum random bits.*

Of course, quantum random bits may be imperfect in a *practical setting.* For example, as time goes on, the number of radioactive nuclei in a radioactive substance decreases. A quantum binary random generator may be become biased when the probability of one outcome is not equal to the probability of the other outcome. It is however less of a problem than one might expect at first sight. Post-processing algorithms can be used to remove bias from a sequence of quantum random numbers affected by bias. The simplest unbiasing procedure was first proposed by von Neumann [40].[8] The bits of a sequence are grouped in strings of two bits. The strings 00 and 11 are discarded; the string 01 is replaced by 0 and the string 10 is replaced by 1. After this procedure, the bias is removed from the sequence. The cost of applying an unbiasing procedure to a sequence is that it is shortened; in the case of von Neumann’s procedure, the length of the unbiased sequence will be at most 25% of the length of the raw sequence. Other, more efficient, unbiasing procedures exist. Peres [45] proved that the number of bits produced by iterating von Neumann’s procedure is arbitrarily close to the entropy bound.

[6] The asymptotic time complexity of the algorithm is bounded by $(\log n)^{12} q(\log \log n)$, where q is some polynomial.

[7] See [24] for pitfalls in using traditional pseudo-random number generation techniques and [22] for Nescape error.

[8] Unbiasing is a compression procedure.

Thirdly, another open question is: *Exactly how much more powerful a Turing machine working with "an oracle of quantum random bits" can be?* [9] This "machine" (which is different from the classical probabilistic Turing machine) can, at any time of the computation, ask the "quantum oracle" to supply an arbitrarily long (but finite) quantum random string. It won't have access to an infinite sequence, but (theoretically) to an unbounded finite set of quantum random bit strings.[10] *Can this immense power be exploited?* [11]

A superficial attack suggests that it is unlikely that a Turing machine augmented with a source of quantum random strings will be capable of solving the Halting Problem: even stepping across the Turing barrier is no guarantee of being able to solve the Halting Problem. But what is the Halting Problem, and why it is the "Philosopher's Stone" of computer science?

The Halting Problem is the problem to decide whether an arbitrarily specified Turing machine halts after a finite number of steps for a given input; the problem cannot be solved by any Turing machine, as Turing proved in 1936! Solving this problem would open a huge box of knowledge. For example, assume you want to know whether every even number greater than 3 is the sum of two primes. We can see that $4 = 2+2, 6 = 3+3, 8 = 3+5$, and so on. In fact, this conjecture has been verified computationally for numbers up to several billion. But is it true for all natural numbers?

One can construct a Turing machine that generates the numbers $4, 6, 8, \ldots$ one after another and checks each of them to see whether it has the above property or not. The machine stops when it finds the first number not having the property; otherwise it continues on to the next number. Clearly, knowing whether this machine stops or not answers the original question. Incidentally, this question is one of the oldest unsolved problems in number theory, a question formulated by Goldbach 262 years ago in a letter to Euler (see [37]). Many other problems can be solved in a similar manner, including the famous Riemann Hypothesis (see [19]), as Matiyasevich proved in [38], p. 121–122.[12]

All theoretical proposals for transcending the Turing barrier (see, for example, [25,10,34]) have been challenged on grounds of physical in-feasibility (see [21]): they require infinite time, infinite memory resources (or both), infinite precision measurements, etc. It's ironic that we now have a method that works in the physical world, but one that seems difficult to justify mathematically because it rests on the assumption that quantum processes are genuinely random.

[9] The idea of computing with deterministic chaos was investigated in [49,50].

[10] The insight provided by the refutation (see [35,18]) of Bennett and Gill's "Random Oracle Hypothesis", [3]—which basically states that the relationships between complexity classes which hold for almost all relativized worlds must also hold in the unrelativized case—suggests that random oracles are extremely powerful; contrast this scenario with the behaviour of probabilistic primality tests run with algorithmically random bits, cf. [17].

[11] Related results can be found in [2].

[12] A rough estimation shows that solving the Goldbach Conjecture is equivalent to deciding the halting status of a RAM program of less than 2,000 bits; for Riemann Hypothesis the program will have about 10,000 bits.

The relation between the Riemann Hypothesis and quantum randomness seems to be more profound (see [48] and [8], chapter 11). Maybe it's not a random fact that in both of them as well as in the fast Monte-Carlo simulation, primes play a central role.

5 Complexity and Incompleteness

Gödel's incompleteness theorem states that every finitely-specified consistent theory which is strong enough to include arithmetic is either inconsistent, incomplete or both. Zermelo-Fraenkel set theory with the Axiom of Choice (ZFC) is such a theory.

Gödel's original proof as well as most subsequent proofs are based on the following idea: a theory which is consistent and strong enough can express statements about provability within the theory, statements which are not provable by the theory, but which through a proof by contradiction, turn out to be true. This type of proof of incompleteness does not answer the questions of whether independence (a true and unprovable statement is called independent) is a widespread phenomenon nor which kinds of statements can be expected to be independent.

Recall that we fixed the universal machine U and H denotes H_U.

The first complexity-theoretic version of incompleteness was discovered by Chaitin [13]:

> *Consider a consistent, sound, finitely-specified theory strong enough to formalise arithmetic and denote by $\mathcal{T}$ its set of theorems. Then, there exists a constant M, which depends upon U and $\mathcal{T}$, such that whenever the statement "$H(x) > n$" is in $\mathcal{T}$ we have $n \leq M$.*

As the complexity $H(s)$ is unbounded, each true statement of the form "$H(x) > m$" with $m > M$ (and, of course, there are infinitely many such statements) is unprovable in the theory.

The high H-complexity of the statements "$H(x) > m$" with $m > M$ is a source of their unprovability. Is every true statement s with $H(s) > M$ unprovable by the theory? Unfortunately, the answer is *negative* because only finitely many statements s have complexity $H(s) \leq M$ in contrast with the fact that the set of all theorems of the theory is infinite. For example, ZFC or Peano Arithmetic trivially prove all statements of the form "$n + 1 = 1 + n$", but the H-complexity of the statement "$n + 1 = 1 + n$" grows unbounded with n. Can the "heuristic principle" proposed by Chaitin in [14], p. 69, namely that "a set of axioms of complexity N cannot yield a theorem of complexity substantially greater than N"[13] be rescued?

[13] The best "approximation" of this principle supported by Chaitin's proof in [13] is that "one cannot prove, from a set of axioms, a theorem that is of greater H–complexity than the axioms *and know* that one has done it"; see [14], p. 69.

The answer is affirmative, but to obtain it we need to change again the complexity measure, specifically, we work with the δ–complexity $\delta(x) = H(x) - |x|$ instead of $H(x)$.[14] To quickly understand the difference between H and δ note that the H-complexity of the statements "$n+1 = 1+n$" grows unbounded with n, but the "intuitive complexities" of the statements "$n+1 = 1+n$" remain bounded; this intuition is confirmed by δ–complexity. Of course, a statement with a large δ–complexity has also a large H-complexity, but the converse is not true.

We can now state the complexity-theoretic theorem obtained in Calude and Jürgensen [7]:

Consider a consistent, sound, finitely-specified theory strong enough to formalise arithmetic and denote by $\mathcal{T}$ its set of theorems. Then, there exists a constant N, which depends upon U and $\mathcal{T}$, such that $\mathcal{T}$ does not contain any x with $\delta(x) > N$, i.e., such an x is unprovable in the theory.

The above theorem does not hold true for an arbitrary finitely-specified theory and it is possible to have incomplete theories in which there are no high δ–complexity statements.

Assume now that we have defined in some way the "set of true statements representable in the theory", a set which presumably includes all arithmetical "true statements". Then, the above theorem shows that any "true statement" of δ–complexity higher than N is independent of the theory. On this base we can show (see [7]) that, probabilistically, incompleteness is widespread, thus complementing the result of Calude, Jürgensen, Zimand [9] stating that the set of unprovable statements is topologically large:

Consider a consistent, sound, finitely-specified theory strong enough to formalise arithmetic. The probability that a statement of length n is provable in the theory tends to zero when n tends to infinity, while the probability that a sentence of length n is true is strictly positive.

Using either ∇ or δ we can re-obtain (for proofs see [11,7]) Chaitin's incompleteness result [13] for Omega:

Consider a consistent, sound, finitely-specified theory strong enough to formalise arithmetic. Then, we can effectively compute a constant N such that the theory cannot determine more than N scattered digits of $\Omega_U = 0.\omega_1\omega_2\ldots$

The complexity-theoretic characterisation of the randomness of Ω_U, recast as a "formal uncertainty principle" in terms of the complexity ∇, implies Chaitin's information-theoretic version of incompleteness for Ω_U. This shows that *uncertainty implies algorithmic randomness which, in turn, implies incompleteness.*

[14] It is easy to see that $\delta(x) \approx \log_2 \nabla(x) - |x|$.

In terms of δ–complexity, high complexity is a source of incompleteness which implies that probabilistically incompleteness is not artificial—it's ubiquitous, pervasive.

We can ask ourselves: How large is the constant N in the above theorem? The answer depends on the chosen universal machine U. Indeed, in Calude [6] one proves the following result:

Consider a consistent, sound, finitely-specified theory strong enough to formalise arithmetic. Then, for each universal machine U we can effectively construct a universal machine W such that $\Omega_U = \Omega_W$ such that the theory can determine at most the initial run of 1's in the expansion of $\Omega_U = 0.11\ldots1\mathbf{0}\ldots$.

As soon as the first $\mathbf{0}$ appears, the theory becomes useless. If $\Omega_V < 1/2$, then the binary expansion of Ω_V starts with 0, and so we obtain Solovay's theorem [51]:

Consider a consistent, sound, finitely-specified theory strong enough to formalise arithmetic. There effectively exists a universal machine V such that the theory can determine no digit of Ω_V.

We finally note that a Turing machine working with an "oracle of quantum random bits" will outperform a standard Turing machine in generating mathematical theorems from any given set of axioms. Still, even this machine cannot generate all true statements of arithmetic.

6 Final Comments

Is the question "Why did the electron go through this slit instead of the other one?", as unanswerable as the question "Why the nth bit of Ω_U is zero?"? This is a difficult question and we don't answer it; the paper brings some (pale) light into this rather dark picture. Namely, we showed that uncertainty implies algorithmic randomness which, in turn, implies incompleteness. For the machines C whose halting probabilities Ω_C are computable, one can construct a quantum computer for which the uncertainty relation describes conjugate observables. Therefore, in these particular instances, the uncertainty relation is equivalent to Heisenberg's.

We have also argued that even in case quantum randomness is weaker than algorithmic randomness, still the "Turing machine augmented with a source of quantum random bits" is more powerful than any Turing machine. This suggests a new attack on the Church-Turing Thesis, and the following interesting (from both practical and theoretical points of view) open question: *how much power has this hybrid machine?* Finally, we have showed that high algorithmic complexity (in particular, algorithmic randomness) is a source of incompleteness, which is pervasive because randomness is ubiquitous.

Acknowledgment

I am much indebted to the chairs of the fourth edition of the conference "Machines, Computation and Universality", Anatoly Beltiukov, Nikolai Kossovskii and Maurice Margenstern, for their invitation to give this talk. I am very grateful to John Casti, Greg Chaitin, Tien Kieu, David Oliver, Mike Stay, Karl Svozil and Garry Tee for illuminating discussions.

References

1. M. Agrawal, N. Kayal, N. Saxena. PRIMES is in P, `http://www.cse.iitk.ac.in/primality.pdf`, 6 August 2002.
2. E. Allender, H. Buhrman, M. Koucký. What can be efficiently reduced to the Kolmogorov-random strings? *Electronic Colloquium on Computational Complexity, Report 44*, 2004, 19 pp.
3. C. H. Bennett, J. Gill. Relative to a random oracle $A, P^A \neq NP^A \neq \text{co-}NP^A$ with probability 1, *SIAM Journal on Computing* 10, 1(1981), 96–113.
4. D. J. Berkeland, D. A. Raymondson, V. M. Tassin. Tests for non-randomness in quantum jumps, Los Alamos preprint archive, `http://arxiv.org/abs/physics/0304013`, 2 April 2004.
5. C. S. Calude. *Information and Randomness. An Algorithmic Perspective*, Springer Verlag, Berlin, 2nd Edition, Revised and Extended, 2002.
6. C. S. Calude. Chaitin Ω numbers, Solovay machines and incompleteness, *Theoret. Comput. Sci.* 284 (2002), 269–277.
7. C. S. Calude, H. Jürgensen. Is Complexity a Source of Incompleteness?, *CDMTCS Research Report* 241, 2004, 15 pp. Los Alamos preprint archive, `http://arxiv.org/abs/math.LO/0408144`, 11 August 2004, 12 pp.
8. C. Calude, P. Hertling, B. Khoussainov. Do the zeros of Riemann's zeta–function form a random sequence? *Bull. Eur. Assoc. Theor. Comput. Sci. EATCS* 62 (1997), 199–207.
9. C. Calude, H. Jürgensen, M. Zimand. Is independence an exception? *Appl. Math. Comput.* 66 (1994), 63–76.
10. C. S. Calude, B. Pavlov. Coins, quantum measurements, and Turing's barrier, *Quantum Information Processing* 1, 1–2 (2002), 107–127.
11. C. S. Calude, M. A. Stay. From Heinsenberg to Gödel via Chaitin, *International Journal of Theoretical Physics*, accepted. E-print as *CDMTCS Research Report* 235, 2004, 15 pp. and Los Alamos preprint archive, `http://arXiv:quant-ph/0402197`, 26 February 2004.
12. C. Calude, M. Zimand. A relation between correctness and randomness in the computation of probabilistic algorithms, *Internat. J. Comput. Math.* 16 (1984), 47–53.
13. G. J. Chaitin. A theory of program size formally identical to information theory, *J. Assoc. Comput. Mach.* 22 (1975), 329–340.
14. G. J. Chaitin. *Information–Theoretic Incompleteness*, World Scientific, Singapore, 1992.
15. G. J. Chaitin. *Leibniz, Information, Math and Physics*, `http://www.cs.auckland.ac.nz/CDMTCS/chaitin/kirchberg.html.`
16. G. J. Chaitin. *META MATH! The Quest for Omega*, Pantheon Books, New York, 2005 (to appear).

17. G. J. Chaitin, J. T. Schwartz. A note on Monte-Carlo primality tests and algorithmic information theory, *Comm. Pure Appl. Math.* 31(1978), 521–527.
18. R. Chang, B. Chor, O. Goldreich, J. Hartmanis, J. Hastad, D. Ranjan, P. Rohatgi. The random oracle hypothesis is false, *J. Comput. System Sci.* 49, 1 (1994), 24–39.
19. `http://www.claymath.org/millennium/Riemann_Hypothesis/`.
20. M. Davis. *The Universal Computer: The Road from Leibniz to Turing*, Norton, New York, 2000.
21. M. Davis. The myth of hypercomputation, in C. Teuscher (ed.). *Alan Turing: Life and Legacy of a Great Thinker*, Springer-Verlag, Heidelberg, 2003, 195–211.
22. J-P. Delahaye. *L'Intelligence and le Calcul*, BELIN, Pour la Science, Paris, 2002.
23. R. Downey, D. Hirschfeldt. *Algorithmic Randomness and Complexity*, Springer-Verlag, Heidelberg, 2005 (to appear).
24. D. E. Eastlake 3rd, S. Crocker, J. Schiller, *Randomness Recommendations for Security*, RFC 1750, December 1994, 30 pp.
25. G. Etesi, I. Németi. Non-Turing computations via Malament-Hogarth space-times, *International Journal of Theoretical Physics* 41 (2002), 341-370.
26. R. P. Feynman. Simulating physics with computers, *International Journal of Theoretical Physics* 21 (1982), 467–488.
27. `http://www.fourmilab.ch/hotbits/`.
28. D. I. Golenko. Generation of uniformly distributed random variables on electronic computers, in Yu. A. Shreider (ed.), translated from Russian by G. J. Tee. *The Monte Carlo Method: The Method Statistical Trials*, Pergamon Press, Oxford, 1966, 257–305.
29. W. Heisenberg. Über den Anschaulichen Inhalt der Quantentheoretischen Kinematik und Mechanik, *Zeitschrift für Physik* 43 (1927), 172–198. English translation in J. A. Wheeler, H. Zurek (eds.). *Quantum Theory and Measurement*, Princeton Univ. Press, Princeton, 1983, 62–84.
30. `http://www.idquantique.com/`.
31. `http://www.idquantique.com/img/QuantisBoth.jpg`.
32. T. Jennewein, U. Achleitner, G. Weihs, H. Weinfurter, A. Zeilinger. A fast and compact quantum random number generator, *Rev. Sci. Instr.* 71 (2000), 1675–1680.
33. E. H. Kennard. Zur Quantenmechanik einfacher Bewegungstypen, *Zeitschrift für Physik* 44 (1927), 326–352.
34. T. D. Kieu. Computing the non-computable, *Contemporary Physics* 44, 1 (2003), 51–71.
35. S. A. Kurtz. On the random oracle hypothesis, *Information and Control* 57, 1 (1983), 40–47.
36. G. W. Leibniz. *Discours de métaphysique*, Gallimard, Paris, 1995.
37. `http://www.mathstat.dal.ca/~joerg/pic/g-letter.jpg`.
38. Yu. V. Matiyasevich. *Hilbert's Tenth Problem*, MIT Press, Cambridge, MA, 1993.
39. G. Milburn. *The Feynman Processor. An Introduction to Quantum Computation*, Allen & Unwin, St. Leonards, 1998.
40. J. von Neumann. Various techniques used in connection with random digits, *National Bureau of Standards Applied Mathematics Series* 12 (1951), 36–38.
41. D. Oliver. Email to C. Calude, 20 August 2004.
42. `http://www.randomnumbers.info`.
43. G. L. Miller. Riemann's hypothesis and tests of primality, *J. Comput. System Sci.* 13 (1976), 300–317.
44. A. Peres. *Quantum Theory: Concepts and Methods*, Kluwer Academic Publishers, Dordrecht, 1993.

45. Y. Peres. Iterating von Neumann's procedure for extracting random bits, *Ann. Stat.* 20 (1992), 590–597.
46. M. O. Rabin. Probabilistic algorithms, in J. F. Traub (ed.). *Algorithms and Complexity, New Directions and Recent Results*, Academic Press, New York, 1976, 21–39.
47. *A Million Random Digits with 100,000 Normal Deviates*, The RAND Corporation, The Free Press, Glencoe, IL, 1955; online edition: `http://www.rand.org/publications/classics/randomdigits/`.
48. M. du Sautoy. *The Music of the Primes*, HarperCollins, New York, 2003,
49. S. Sinha, W. L. Ditto. Dynamics based computation, *Physical Letters Review* 81, 10 (1998), 2156–2159.
50. S. Sinha, W. L. Ditto. Computing with distributed chaos, *Physical Review E* 60, 1 (1999), 363–377.
51. R. M. Solovay. A version of Ω for which ZFC can not predict a single bit, in C. S. Calude, G. Păun (eds.). *Finite Versus Infinite. Contributions to an Eternal Dilemma*, Springer-Verlag, London, 2000, 323–334.
52. K. Svozil. The quantum coin toss-testing microphysical undecidability, *Physics Letters* A143, 433–437.
53. K. Svozil. *Randomness & Undecidability in Physics*, World Scientific, Singapore, 1993.
54. K. Tadaki. Upper bound by Kolmogorov complexity for the probability in computable POVM measurement, Los Alamos preprint archive, `http://arXiv:quant-ph/0212071`, 11 December 2002.
55. S. Wolfram. Statistical mechanics of cellular automata, *Reviews of Modern Physics* 55 (1983), 601–644.
56. S. Wolfram. *A New Kind of Science*, Wolfram Media, Champaign, IL, 2002.
57. U. Yurtsever. Quantum mechanics and algorithmic randomness, *Complexity* 6, 1 (2002), 27–31.

On the Complexity of Universal Programs

Alain Colmerauer

Laboratoire d'Informatique Fondamentale de Marseille,
CNRS et Universités de Provence et de la Méditerranée

Abstract. This paper provides a framework enabling to define and determine the complexity of various universal programs U for various machines. The approach consists of first defining the complexity as the average number of instructions to be executed by U, when simulating the execution of one instruction of a program P with input x.
To obtain a complexity that does not depend on P or x, we then introduce the concept of an introspection coefficient expressing the average number of instructions executed by U, for simulating the execution of one of its own instructions. We show how to obtain this coefficient by computing a square matrix whose elements are numbers of executed instructions when running selected parts of U on selected data. The coefficient then becomes the greatest eigenvalue of the matrix.
We illustrate the approach using two examples of particularly efficient universal programs: one for a three-symbol Turing Machine (blank symbol not included) with an introspection coefficient of 3 672.98, the other for an indirect addressing arithmetic machine with an introspection coefficient of 26.27.

1 Introduction

For the past several years I have been teaching an introductory course designed to initiate undergraduate students to low level programming. My approach was to start teaching them how to program Turing machines. The main exercise in the course consisted of completing and testing a universal program whose architecture I provided. The results were disappointing, the universal program being too slow for executing sizeable programs. Among others it was impossible to run the machine on its own code, in the sense explained in section 4. In subsequent years, I succeeded in designing considerably more efficient universal programs, even though they became increasingly more complex. These improved programs were capable to execute their own code in reasonable times. For simulating the execution of one of its own instructions, the last program executes an average number of 3 672.98 instructions, or more exactly its introspection coefficient – a key concept introduced in this paper – is equal to 3 672.98.

This paper presents this result in a more general context concerning machines other then Turing machines. It is organized in 5 sections followed by an appendix. The first constitutes this introduction and the last the conclusion. Section 2 introduces the concepts of programmed machine, machine, program, transition and instruction. Section 3 illustrates those concepts on three examples: Turing machines, Turing machines with internal direction (a useful variant of the previous machines), and an indirect addressing arithmetic machine. In Section 4, the main component of this paper, it is shown how to check the existence of introspection coefficients and how to compute their values. The proof of the theorem presented in that section is fairly lengthy. As most proofs it is omitted by lack of space. A more complete version of the paper, with all proofs, is under preparation. Sections 5 and 6 are devoted to two specially efficient universal programs: the first one, as already mentioned, for a Turing machine, the second one for an indirect addressing arithmetic machine.

We are not aware of other work on the design of efficient universal programs. Let us however mention the well known contributions of M. Minsky [1] and Y. Rogozin [3] in the design of universal programs for Turing machines with very small numbers of states. Surprisingly, they seem particularly inefficient in terms of number of executed instructions.

2 Machines

2.1 Basic Definitions

Definition 1 A *programmed machine* is an ordered pair $(\mathcal{M}, P)$, where $\mathcal{M}$ is a *machine* and P a *program* for $\mathcal{M}$.

Definition 2 A *machine* $\mathcal{M}$ is a 5-tuple $(\Sigma, C, \alpha, \omega, \mathcal{I})$, where

Σ, *the alphabet* of $\mathcal{M}$, is a finite not empty set;
C, is a set, generally infinite, of *configurations*; the ordered pairs (c, c') of elements of C are called *transitions*;
α, *the input function*, maps each element x of $\Sigma^\star$ to a configuration $\alpha(x)$;
ω, *the ouput function*, maps each configuration c to an element $\omega(c)$ of $\Sigma^\star$;
$\mathcal{I}$, is a countable set of *instructions*, an *intruction* being a set of *compatibles* transitions, i.e., whose first components are all distinct.

Definition 3 A *program* P for a machine $\mathcal{M}$ is a finite subset of the instructions set $\mathcal{I}$ of $\mathcal{M}$, such that the transitions of $\bigcup P$ are compatible.[1]A configuration c of $\mathcal{M}$ is *final for* P, if there exists no transition of $\bigcup P$ starting with c.

2.2 How a Machine Operates

Let $\mathcal{M} = (\Sigma, C, \alpha, \omega, \mathcal{I})$ be a machine and P a program for $\mathcal{M}$. The operation of the machine $(\mathcal{M}, P)$ is explained by the diagram:

$$\begin{tikzcd}
x \arrow[d] & & & & & y \\
c_0 \arrow[r] & c_1 \arrow[r] & c_2 & \cdots & c_{n-1} \arrow[r] & c_n \arrow[u]
\end{tikzcd}$$

[1] P being a set of sets $\bigcup P$ denotes the set of elements which are member of at least one element of P and thus the set of transitions involved in program P.

and more precisely by the definition of the following functions[2], where x is a word on Σ and c, c' configurations of C:

$$\mathit{orbit}_{\mathcal{M}}(P,\, x) = \begin{cases} \text{the longest sequence } c_0, c_1, c_2, \ldots \text{ with} \\ c_0 = \alpha(x) \text{ and each } (c_i, c_{i+1}) \text{ an element of } \bigcup P. \end{cases}$$

$$\mathit{out}_{\mathcal{M}}(P,\, x) = \begin{cases} \nearrow, & \text{if } \mathit{orbit}(P,\, x) \text{ is infinite,} \\ \omega(c_n), & \text{if } \mathit{orbit}(P,\, x) \text{ ends with } c_n \end{cases}$$

and, for dealing with complexity,

$$\mathit{track}_{\mathcal{M}}(P,\, c,\, c') = \begin{cases} \text{the shortest sequence of the form} \\ (c_0, c_1)\,(c_1, c_2)\,(c_2, c_3)\,\ldots\,(c_{n-1}, c_n), \\ \text{with } c = c_0, \text{ each } (c_i, c_{i+1}) \in \bigcup P,\ c_n = c', \text{ if it exists,} \\ \nearrow, \text{ if this shortest sequence does not exist,} \end{cases}$$

$$\mathit{track}_{\mathcal{M}}(P,\, x) = \begin{cases} \text{the longest sequence of the form} \\ (c_0, c_1)\,(c_1, c_2)\,(c_2, c_3)\,\ldots \\ \text{with } \mathit{orbit}(P,\, x) = c_0, c_1, c_2, \ldots. \end{cases}$$

$$\mathit{cost}_{\mathcal{M}}(P,\, x) = \begin{cases} \infty, & \text{if } \mathit{track}(P,\, x) \text{ is infinite,} \\ |\mathit{track}(P,\, x)|, & \text{if } \mathit{track}(P,\, x) \text{ is finite,} \end{cases}$$

where $|\mathit{track}(P,\, x)|$ denotes the length of the finite sequence $\mathit{track}(P,\, x)$.

2.3 Simulation of a Machine by Another

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two machines of same alphabet Σ and let $\mathcal{P}_1$ and $\mathcal{P}_2$ be the sets of their programs.

Definition 4 Two programmed machines of the form $(\mathcal{M}_1, P_1)$ and $(\mathcal{M}_2, P_2)$ are *equivalent* if, for all $x \in \Sigma^\star$,

$$\begin{aligned} \mathit{out}_{\mathcal{M}_1}(P_1,\, x) &= \mathit{out}_{\mathcal{M}_2}(P_2,\, x), \\ \mathit{cost}_{\mathcal{M}_1}(P_1,\, x) &= \mathit{cost}_{\mathcal{M}_2}(P_2,\, x). \end{aligned}$$

Definition 5 The program transformation $f_1 : \mathcal{P}_1 \to \mathcal{P}_2$ *simulates* $\mathcal{M}_1$ on $\mathcal{M}_2$ if, for all $P_1 \in \mathcal{P}_1$, the programmed machines $(\mathcal{M}_1, P_1)$ and $(\mathcal{M}_2, f_1(P_1))$ are equivalent.

[2] Index $\mathcal{M}$ is omitted when there is no ambiguity.

3 Examples of Machines

3.1 Turing Machines

Informally these are classical Turing machines with a bi-infinite tape and instructions written $[q_i, abd, q_j]$, with $d = L$ or $d = R$, meaning : if the machine is in state q_i and the symbol read by the read-write head is a, the machine replaces a by b, then moves its head one symbol to the left or the right, depending whether $d = L$ or $d = R$, and change its state to q_j. Initially the entire tape is filled with blanks except for a finite portion which contains the initial input, the read-write head being positioned on the symbol which precedes this input. When there are no more instructions to be executed the machine output the longest word which contains no blank symbols and which starts just after the position of the read-write head.

Formally one first introduces an infinite countable set $\{q_1, q_2, \ldots\}$ of *states* and a special symbol $\mathtt{u}$, *the blank*. For any alphabet word x on an alphabet of the form $\Sigma \cup \{\mathtt{u}\}$, one writes $\cdot x$ for x, with all its beginning blanks erased, and $x\cdot$ for x, with all its ending blanks erased.

Definition 6 A *Turing machine* has a 5-tuple of the form $(\Sigma, C, \alpha, \omega, \mathcal{I})$ where,

- $\Sigma \neq \Sigma_{\mathtt{u}}$, with $\Sigma_{\mathtt{u}} = \Sigma \cup \{\mathtt{u}\}$,
- C is the set of 4-tuples of the form $[q_i, \cdot x, a, y\cdot]$, with q_i being any state, x, y taken from $\Sigma_{\mathtt{u}}^\star$ and a taken from $\Sigma_{\mathtt{u}}$,
- $\alpha(x) = [q_1, \varepsilon, \mathtt{u}, x]$, for all $x \in \Sigma^\star$,
- $\omega([q_i, \cdot x, a, y\cdot])$ is the longest element of $\Sigma^\star$ beginning $y\cdot$,
- $\mathcal{I}$ is the set of instructions denoted and defined, for all sates q_i, q_j and elements a, b of $\Sigma_{\mathtt{u}}$, by
  $[q_i,\, abL,\, q_j] \overset{\text{def}}{=} \{([q_i, \cdot xc, a, y\cdot],\, [q_j, \cdot x, c, by\cdot]) \mid (x, c, y) \in E\}$,
  $[q_i,\, abR,\, q_j] \overset{\text{def}}{=} \{([q_i, \cdot x, a, cy\cdot],\, [q_j, \cdot xb, c, y\cdot]) \mid (x, c, y) \in E\}$,
  with $E = \Sigma_{\mathtt{u}}^\star \times \Sigma_{\mathtt{u}} \times \Sigma_{\mathtt{u}}^\star$.

3.2 Turing Machines with Internal Direction

These are a variant of the above described Turing machines with an internal moving head direction whose initial value is equal to left-right. The instructions are written $[q_i,\, abs,\, q_j]$, with $s = +$ or $s = -$, meaning : if the machine is in state q_i and the symbol read by the read-write head is a, the machine replaces a by b, keeps its internal direction or changes it depending whether $s = +$ or $s = -$, moves its read-write head one symbol in the new internal direction, and changes its states to q_j.

Formally we get:

Definition 7 A *Turing machine with internal direction* has a 5-tuple of the form $(\Sigma, C, \alpha, \omega, \mathcal{I})$ where,

- $\Sigma \neq \Sigma_{\mathtt{u}}$, with $\Sigma_{\mathtt{u}} = \Sigma \cup \{\mathtt{u}\}$,
- C is the set of 5-tuples of the form $[d, q_i, \cdot x, a, y\cdot]$, with $d \in \{L, R\}$, q_i being a state, x, y taken from $\Sigma_{\mathtt{u}}^\star$ and a taken from $\Sigma_{\mathtt{u}}$,

- $\alpha(x) = [R, q_1, \varepsilon, \mathtt{u}, x]$, for all $x \in \Sigma^\star$,
- $\omega([d, q_i, \cdot x, a, y\cdot])$ is the longest element of $\Sigma^\star$ beginning $y\cdot$,
- $\mathcal{I}$ is the set of instruction denoted and defined, for all states q_i, q_j, all elements a, b of $\Sigma_{\mathtt{u}}$ and all $s \in \{+, -\}$, by
  $[q_i,\ abs,\ q_j] \overset{\text{def}}{=}$
  $\{([d, q_i, \cdot xc, a, y\cdot],\ [L, q_j, \cdot x, c, by\cdot]) \mid\ (d, s) \in E_1$ and $(x, c, y) \in F\} \cup$
  $\{([d, q_i, \cdot x, a, cy\cdot],\ [R, q_j, \cdot xb, c, y\cdot]) \mid\ (d, s) \in E_2$ and $(x, c, y) \in F\}$,
  with $E_1 = \{(L, +), (R, -)\}$, $E_2 = \{(R, +), (L, -)\}$ and $F = \Sigma_{\mathtt{u}}^\star \times \Sigma_{\mathtt{u}} \times \Sigma_{\mathtt{u}}^\star$.

3.3 Relationship Between the Two Types of Turing Machines

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be machines of same alphabet Σ, the first one of type Turing and the second one of type Turing with internal direction, and let $\mathcal{P}_1$ and $\mathcal{P}_2$ denote the sets of their programs. As we will see, there exists a strong relation between these two types of machines. First it can be shown that:

Property 1 *The program transformations $g_1 : \mathcal{P}_1 \to \mathcal{P}_2$ and $g_2 : \mathcal{P}_2 \to \mathcal{P}_1$, defined below, simulate $\mathcal{M}_1$ on $\mathcal{M}_2$ and $\mathcal{M}_2$ on $\mathcal{M}_1$:*

$$g_1(P_1) \overset{def}{=} \{[q_{2i-h},\ abd,\ q_{2j-k}] \mid [q_i,\ abs,\ q_j] \in P' \text{ and } (h, k, d, s) \in E\},$$

$$g_2(P_2) \overset{def}{=} \{[q_{2i-h},\ abs,\ q_{2j-k}] \mid [q_i,\ abd,\ q_j] \in P \text{ and } (h, k, s, d) \in E\},$$

with

$$E = \begin{Bmatrix} (1,\ 1,\ +, R) \\ (1,\ 0,\ -, L) \end{Bmatrix} \cup \begin{Bmatrix} (0,\ 1,\ -, R) \\ (0,\ 0,\ +, L) \end{Bmatrix}.$$

Let us now introduce the following concepts concerning a program P_ℓ for $\mathcal{M}_\ell$, with $\ell = 1$ or $\ell = 2$.

Definition 8 A state q_i is *reachable in P_ℓ*, if $i = 1$ or if there exists a subset of P_ℓ of the form

$$\{[q_{k_0},\ a_1b_1e_1,\ q_{k_1}], [q_{k_1},\ a_2b_2e_2,\ q_{k_2}], \ldots, [q_{k_{m-1}},\ a_mb_me_m,\ q_{k_m}]\},$$

with $k_0 = 1$ ank $k_m = i$, the a_i's and b_i's being taken from Σ and the e_i's being taken from $\{L, R\}$ or $\{+, -\}$, depending on whether $\ell = 1$ or $\ell = 2$.

Definition 9 For all subsets of P_ℓ of the form

$$\{[q_{k_0},\ a_1b_1e_1,\ q_{k_1}], [q_{k_1},\ a_2b_2e_2,\ q_{k_2}], \ldots, [q_{k_{m-1}},\ a_mb_me_m,\ q_{k_m}]\},$$

with the a_i's and b_i's taken from Σ and the e_i's from $\{L, R\}$ or $\{+, -\}$, depending whether $\ell = 1$ or $\ell = 2$, the sequence of 3-tuples $a_1b_1e_1, a_2b_2e_2, \ldots, a_mb_me_m$ is called *a potential effect* of state q_{k_0}. Two states occurring in P which have the same set of potential effects are said to be *mergeable.*

Definitions 10

- *reachable*(P_ℓ) *denotes the set of instructions of* P_ℓ *involving reachable states of* P_ℓ,
- *merged*(P_ℓ) *denotes the program obtained by replacing in* P*, and in parallel, every occurrence of a state* q_i *by the state* q_j *of smallest index such that* q_i *and* q_j *are mergeable,*
- *compact*(P_ℓ) *denotes the program obtained by renaming each state* q_i *of* P_ℓ *by* $q_{i'}$ *in such a way that the integers* i' *are as small as possible and that* $i < j$ *entails* $i' < j'$,
- $\mathit{clean}(P_\ell) = \mathit{compact}(\mathit{reachable}(\mathit{merged}(P_\ell)))$.

One proves that for $\ell = 1$ and $\ell = 2$:

Property 2 *The programmed machines* $(\mathcal{M}_\ell, \mathit{clean}(P_\ell))$ *and* $(\mathcal{M}_\ell, P_\ell)$ *are equivalent.*

One also proves that:

Theorem 1 *The program transformations* $f_1 : \mathcal{P}_1 \to \mathcal{P}_2$ *and* $f_2 : \mathcal{P}_2 \to \mathcal{P}_1$, *defined by* $f = \mathit{clean} \circ g_\ell$, *with* g_ℓ *defined in Property 1, simulate* $\mathcal{M}_1$ *on* $\mathcal{M}_2$ *and* $\mathcal{M}_2$ *on* $\mathcal{M}_1$ *and are such that* $f_2 \circ f_1 \circ f_2 \circ f_1 = f_2 \circ f_1$ *and* $f_1 \circ f_2 \circ f_1 \circ f_2 = f_1 \circ f_2$.

3.4 Indirect Addressing Arithmetic Machine

This is a machine with an infinity of registers $r_0, r_1, r_2, \ldots$. Each register contains an unbounded natural integer. Each instruction starts with a number and the machine always executes the instruction whose number is contained in r_0 and, except in one case, increases r_0 by 1. There are five types of instructions: assigning a constant to a register, addition and subtraction of a register to/from another, two types of indirect assignment of a register to another and zero-testing of a register content.

More precisely and in accordance with our definition of a machine:

Definition 11 An indirect addressing arithmetic machine has a 5-tuple of the form $(\Sigma, C, \alpha, \omega, \mathcal{I})$, where,

- $\Sigma = \{c_1, \ldots, c_m\}$,
- C is the set of infinite sequences $r = (r_0, r_1, r_2, \ldots)$ of natural integers,
- $\alpha(a_1 \ldots a_n) = (0, 25, 1, \ldots, 1, r_{24+1}, \ldots, r_{24+n}, 0, 0, \ldots)$, with r_{24+i} equal to $1, \ldots, m$ depending whether a_i equals $c_1, \ldots, c_m$,
- $\omega(r_0, r_1, \ldots) = a_1 \ldots a_n$, with a_i equal to $c_1, \ldots, c_m$ depending whether $\overline{r_{r_1+i}}$ equals $1, \ldots, m$, and n being is the greatest integer such that $r_{r_1}, \ldots, r_{r_1+n}$ are elements of $\{1, \ldots, m\}$,
- $\mathcal{I}$ is the set of instructions denoted and defined, for all natural integers i, j, k, by:

$$\begin{aligned}
[i, \mathit{cst}, j, k] &\overset{\text{def}}{=} \{(r,\, s') \in C^2 \,|\, r_0 = 1,\, s_j = k \text{ and } s_i = r_i \text{ elsewhere}\},\\
[i, \mathit{plus}, j, k] &\overset{\text{def}}{=} \{(r,\, s') \in C^2 \,|\, r_0 = 1,\, s_j = r_j{+}r_k \text{ and } s_i = r_i \text{ elsewhere}\},\\
[i, \mathit{sub}, j, k] &\overset{\text{def}}{=} \{(r,\, s') \in C^2 \,|\, r_0 = 1,\, s_j = r_j{\dot{-}}r_k \text{ and } s_i = r_i \text{ elsewhere}\},\\
[i, \mathit{from}, j, k] &\overset{\text{def}}{=} \{(r,\, s') \in C^2 \,|\, r_0 = 1,\, s_j = r_{r_k} \text{ and } s_i = r_i \text{ elsewhere}\},\\
[i, \mathit{to}, j, k] &\overset{\text{def}}{=} \{(r,\, s') \in C^2 \,|\, r_0 = 1,\, s_{r_j} = r_k \text{ and } s_i = r_i \text{ elsewhere}\},\\
[i, \mathit{ifze}, j, k] &\overset{\text{def}}{=} \{(r,\, s') \in C^2 \,|\, r_0 = 1,\, s_0 = \begin{cases} r_k, \text{if } r_j = 0,\\ r_0, \text{if } r_j \neq 0 \end{cases} \text{and } s_i = r_i \text{ elw.}\},
\end{aligned}$$

with s' equal to s except that $s'_0 = s_0 + 1$ and with $r_j{\dot{-}}r_k = \max\{0, r_j{-}r_k\}$.

4 Universal Programs and Codings

4.1 Introduction

Let $\mathcal{M} = (\Sigma, C, \alpha, \omega, \mathcal{I})$ be a machine and let us code each program P for $\mathcal{M}$ by a word $\mathit{code}(P)$ on Σ.

Definition 12 The pair $(U,\, \mathit{code})$, the program U and the coding function *code*, are said to be *universal* for $\mathcal{M}$, if, for all programs P of $\mathcal{M}$ and for all $x \in \Sigma^\star$,

$$\boxed{\mathit{out}(U,\, \mathit{code}(P) \cdot x) = \mathit{out}(P,\, x)}. \tag{1}$$

If in the above formula we replace P by U, and x by $\mathit{code}(U)^n \cdot x$ we obtain:

$$\mathit{out}(U,\, \mathit{code}(U)^{n+1} \cdot x) = \mathit{out}(U,\, \mathit{code}(U)^n \cdot x)$$

and thus:

Property 3 *If $(U,\, \mathit{code})$ is a universal pair, then for all $n \geq 0$ and $x \in \Sigma^\star$,*

$$\boxed{\mathit{out}(U,\, \mathit{code}(U)^n \cdot x) = \mathit{out}(U,\, x)}. \tag{2}$$

4.2 Complexity and Introspection Coefficient

Let (U, code) be a universal pair for the machine $\mathcal{M} = (\Sigma, C, \alpha, \omega, \mathcal{I})$. The *complexity* of this pair is the average number of transitions performed by U for producing the same effect as a transition of the program P occurring in the input of U. More precisely:

Definition 13 Given a program P for $\mathcal{M}$ and a word x on Σ with $\mathit{cost}(P,\, x) \neq \infty$, the *complexity* of $(U,\, \mathit{code})$ is the real number denoted and defined by

$$\boxed{\lambda(P,\, x) = \frac{\mathit{cost}(U,\, \mathit{code}(P) \cdot x)}{\mathit{cost}(P,\, x)}.}$$

The disavantage of this definition is that the complexity depends on the input of U. For an intrinsic complexity, independ of the input of U, we introduce the *introspection coefficient* of $(U, code)$ whose definition is justified by Property 2:

Definition 14 If for all $x \in \Sigma^\star$, with $cost(U, x) \neq \infty$, the real number

$$\boxed{\lim_{n\to\infty} \lambda(U,\ code(U)^n \cdot x) = \lim_{n\to\infty} \frac{cost(U, code(U)^{n+1} \cdot x)}{cost(U, code(U)^n \cdot x)}}$$

exists and does not depend on x, then this real number is the *introspection coefficient* of the universal pair $(U, code)$.

4.3 Keeping the Same Complexity and Introspection Coefficient

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two machines, with same alphabet Σ, let $\mathcal{P}_1$ and $\mathcal{P}_2$ be the sets of their programms, let $f_1 : \mathcal{P}_1 \to \mathcal{P}_2$ and $f_2 : \mathcal{P}_2 \to \mathcal{P}_1$ be to program transformation and let $(U_2, code_2)$ be a universal pair for $\mathcal{M}_2$. Let us make the following hypothesis:

Hypothesis 1 *The transformation f_1 simulates $\mathcal{M}_1$ on $\mathcal{M}_2$, the transformation f_2 simulates $\mathcal{M}_2$ on $\mathcal{M}_1$ and $code_2 \circ f_1 \circ f_2 = code_2$.*

From Theorem 1 at page 23 it follows:

Property 4 *In the particular case where $\mathcal{M}_1$ is a Turing machine and $\mathcal{M}_2$ a Turing with internal direction, Hypothesis 1 is satisfied by taking the transformations f_1 and f_2 of Theorem 1 and by taking $code_2$ of the form $code_2' \circ f_1 \circ f_2$, with $code_2'$ a mapping of type $\mathcal{P}_2 \to \Sigma^\star$.*

For the sequel, let x, P_1, P_2 respectivley denote an element of $\Sigma^\star, \mathcal{P}_1, \mathcal{P}_2$. We have the following property:

Property 5 *With Hypothesis 1 and if $P_1 = f_2(P_2)$ or $P_2 = f_1(P_1)$, the pair $(U_1,\ code_1)$, with $U_1 = f_2(U_2)$ and $code_1 = code_2 \circ f_1$,*

1. *is universal for $\mathcal{M}_1$,*
2. *has a complexity $\lambda_1(P_1, x)$, undefined or equal to the complexity $\lambda_2(P_2, x)$ of the pair $(U_2, code_2)$, depending whether the real number $\lambda_2(P_2, x)$ is undefined or defined,*[3]
3. *admits the same introspection coefficient as $(U_2, code_2)$ or does not admit an introspection coefficient, depending whether $(U_2, code_2)$ admits or does not admit one,*
4. *is such that $code_1(P_1) = code_2(P_2)$.*

[3] Of course $\lambda_i(P_i, x) = \frac{cost_{\mathcal{M}_i}(U_i,\ code_i(P_i) \cdot x)}{cost_{\mathcal{M}_i}(P_i, x)}$.

Proof Let us first prove claim 4. If $P_2 = f_1(P_1)$, since $code1 = code_2 \circ f_1$, we have

$$code_1(P_1) = code_2(f_1(P_1)) = code(P_2).$$

If $P_1 = f_2(P_2)$, since $code_2 = code_2 \circ f_1 \circ f_2$ and $code_2 \circ f_1 = code_1$, we have $code_2 = code_1 \circ f_2$ and thus

$$code_2(P_2) = code_1(f_2(P_2)) = code_1(P_1).$$

Claim 1 is proven by the equalities below, where Q_1 is any element of $\mathcal{P}_1$:

$$\begin{array}{ll}
out_{\mathcal{M}_1}(Q_1, x) \;= & \text{(since } f_1 \text{ simulates } \mathcal{M}_1 \text{ on } \mathcal{M}_2)\\
out_{\mathcal{M}_2}(f_1(Q_1), x) \;= & \text{(since } (U_2, code_2) \text{ is universal for } \mathcal{M}_2)\\
out_{\mathcal{M}_2}(U_2,\ code_2(f_1(Q_1)) \cdot x) \;= & \text{(since } f_2 \text{ simulates } \mathcal{M}_2 \text{ on } \mathcal{M}_1)\\
out_{\mathcal{M}_1}(f_2(U_2), code_2(f_1(Q_1)) \cdot x) \;= & \text{(since } f_2(U_2) = U_1 \text{ and } code_2 \circ f_1 = code_1)\\
out_{\mathcal{M}_1}(U_1,\ code_1(Q_1) \cdot x). &
\end{array}$$

Claim 2 is proven by showing the equality of pairs

$$\begin{bmatrix} cost_{\mathcal{M}_1}(U_1,\ code_1(P_1) \cdot x) \\ cost_{\mathcal{M}_1}(P_1,\ x) \end{bmatrix} = \begin{bmatrix} cost_{\mathcal{M}_2}(U_2,\ code_2(P_2) \cdot x) \\ cost_{\mathcal{M}_2}(P_2,\ x) \end{bmatrix}. \tag{3}$$

First of all,

$$\begin{array}{ll}
cost_{\mathcal{M}_1}(U_1, code_1(P_1) \cdot x) \;= & \text{(since } U_1 = f_2(U_2),\ code_1(P_1) = code_2(P_2))\\
cost_{\mathcal{M}_1}(f_2(U_2), code_2(P_2) \cdot x) \;= & \text{(since } f_2 \text{ simulates } \mathcal{M}_2 \text{ on } \mathcal{M}_1)\\
cost_{\mathcal{M}_2}(U_2, code_2(P_2) \cdot x). &
\end{array}$$

Secondly, since f_2 simulates $\mathcal{M}_2$ on $\mathcal{M}_1$, if $P_1 = f_2(P_2)$, and since f_2 simulates $\mathcal{M}_1$ on $\mathcal{M}_2$, if $P_1 = f_2(P_2)$, we have

$$cost_{\mathcal{M}_1}(P_1, x) = cost_{\mathcal{M}_2}(P_2, x).$$

Finally claim 3 is proven by the sequence of equalities:

$$\begin{array}{ll}
\begin{bmatrix} cost_{\mathcal{M}_1}(U_1,\ code_1(U_1) \cdot code_1(U_1)^n \cdot x) \\ cost_{\mathcal{M}_1}(U_1,\ code_1(U_1)^n \cdot x) \end{bmatrix} & = \begin{array}{l}\text{(by replacing } P_1 \text{ by } U_1 \text{ and}\\ x \text{ by } code_1(U_1)^n \cdot x \text{ in (3))}\end{array}\\
\begin{bmatrix} cost_{\mathcal{M}_2}(U_2,\ code_2(U_2) \cdot code_1(U_1)^n \cdot x) \\ cost_{\mathcal{M}_1}(U_2,\ code_1(U_1)^n \cdot x) \end{bmatrix} & = \text{(since } code_1(U_1) = code_2(U_2))\\
\begin{bmatrix} cost_{\mathcal{M}_2}(U_2,\ code_2(U_2) \cdot code_2(U_2)^n \cdot x) \\ cost_{\mathcal{M}_2}(U_2,\ code_2(U_2)^n \cdot x) \end{bmatrix}. &
\end{array}$$

This completes the proof.

4.4 Existence and Value of the Introspection Coefficient

Let $(U, code)$ be a universal pair for a machine $\mathcal{M} = (\Sigma, C, \alpha, \omega, \mathcal{I})$. Given a word x on Σ, we assume that the computation of the word y by $y = out(U,\ x)$ can be synchronized with the computation of the same word y by $y = out(U,\ code(U)\cdot x)$, according to the following diagram:

$$\begin{tikzcd}
 & x \arrow[d] & & & & & y & \\
code(U)\cdot x \arrow[d] & \bullet \arrow[r, "1"] \arrow[d] & \bullet \arrow[r, "2"] \arrow[d] & \bullet \arrow[r, "2"] \arrow[d] & \bullet \arrow[r, "1"] \arrow[d] & \bullet \arrow[r, "3"] \arrow[d] & \bullet \arrow[u] \arrow[dr] & y \\
\bullet \arrow[r, "1\ 1\ 4"] & \bullet \arrow[r, "3\ 1"] & \bullet \arrow[r, "2\ 3\ 3"] & \bullet \arrow[r, "2\ 3\ 3"] & \bullet \arrow[r, "3\ 1"] & \bullet \arrow[rr, "3\ 3\ 1\ 5\ 5"] & & \bullet \arrow[u]
\end{tikzcd}$$

More precisely we make the hypothesis:

Hypothesis 2 *There exists*

- a synchronization function $\Phi : C_U \to C_U$, *with* $\Phi(c)$ *final for* U *when* c *is final for* U,
- a labelling function $\mu : (\bigcup U)^\star \to (1..n)^\star$ *with* n *a positive integer and* μ *being surjective, such that* $\mu(t_1 \dots t_k) = \mu(t_1)\dots\mu(t_k)$, *for all* $t_1 \dots t_k \in (\bigcup U)^\star$,
- an initial sequence of labels $\delta \in (1..n)^\star$, *independent of* x, *such that*

$$\delta = \mu(track(U,\ \alpha(code(U)\cdot x),\ \Phi(\alpha(x)))),\ \textit{for all } x \in \Sigma^\star,$$

- a label rewriting $\varphi : 1..n \to (1..n)^\star$ *such that, for all* $(c, c') \in \bigcup U$,

$$\varphi(\mu(c, c')) = \mu(track(U,\ \Phi(c),\ \Phi(c'))).$$

We then introduce the column vector B and the square matrix A:

$$B = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}, \qquad b_i = \text{number of occurrences of } i \text{ in } \delta,$$
$$A = \begin{bmatrix} a_{11} \cdots a_{nn} \\ \vdots \qquad \vdots \\ a_{1n} \cdots a_{nn} \end{bmatrix}, \ a_{ij} = \text{number of occurrences of } i \text{ in } \varphi(j). \tag{4}$$

and we conclude by the main theorem of the paper:

Theorem 2 *If the matrix* A *admits a real eigenvalue* λ, *whose multiplicity is equal to* 1 *and whose value is strictly greater than* 1 *and strictly greater then the absolute value* λ' *of the other eigenvalues of* A *then:*

- *the limit matrix* $\bar{A} = \lim_{n\to\infty}(\frac{1}{\lambda}A)^n$ *exists and has at least one non-zero element,*
- *if* $||\bar{A}B|| \neq 0$, *the introspection coefficient of* U *exists and is equal to* λ,
- *if* ℓ *is a real number such that* $\lambda' < \ell < \lambda$ *and the* X_i*'s column vectors defined by* $X_0 = B$ *and* $X_{n+1} = \frac{1}{\ell}AX_n$, *then, when* $n \to \infty$,

$$||X_n|| \to \begin{cases} 0, & \textit{if } ||\bar{A}B|| = 0, \\ \infty, & \textit{if } ||\bar{A}B|| \neq 0. \end{cases}$$

Here $||X||$ denotes the sum of the components of X.

5 Our Universals Pairs for the Two Types of Turing Machines

5.1 Introduction

We now present two particularly efficient universal pairs, (U_1, code_1), (U_2, code_2), one for the Turing machine $\mathcal{M}_1$ with alphabet $\Sigma = \{\mathtt{o}, \mathtt{i}, \mathtt{z}\}$ and the other for the Turing machine with internal direction $\mathcal{M}_2$ and same alphabet Σ.

The pair (U_1, code_1) is built from the pair (U_2, code_2) by taking $U_1 = f_2(U_2)$ and $\mathit{code}_1 = \mathit{code}_2 \circ f_1$, where f_1 and f_2 are the program transformations introduced in Theorem 1 at page 23. Since code_2 will be of the form $\mathit{code}'_2 \circ f_1 \circ f_2$, according to Properties 4 and 5, the pair (U_1, code_1) is indeed universal for $\mathcal{M}_1$ and has the same complexity properties as the pair (U_2, code_2).

Let us mention that U_1 has 361 instructions and 106 states while U_1 has only 184 instructions and 54 states. It remains to present the pair (U_2, code_2).

5.2 Coding Function of the Universal Pair (U_2, code_2)

In order to assign a position to each instruction $[q_i, abs, q_j]$ of a program for $\mathcal{M}_2$, we first introduce the numbers:

$$\pi(i,a) = 4(i-1) + \begin{cases} 1, \text{ if } a = \mathtt{u} \\ 2, \text{ if } a = \mathtt{o} \\ 3, \text{ if } a = \mathtt{i} \\ 4, \text{ if } a = \mathtt{z} \end{cases}, \qquad \pi(i) = \tfrac{1}{2}(f(i,\mathtt{o}) + f(i,\mathtt{i})),$$

defined for any positive integer i and any symbol $a \in \{\mathtt{u}, \mathtt{o}, \mathtt{i}, \mathtt{z}\}$. Then let P_2 be a program for $\mathcal{M}_2$ and let $P'_2 = f_1(f_2(P_2))$. We take $\mathit{code}_2(P_2) = \mathit{code}'_2(P'_2)$, with $\mathit{code}'_2(P'_2)$ the word on $\{\mathtt{o}, \mathtt{i}, \mathtt{z}\}$

$$\mathtt{z}I_{4n}\mathtt{z}\ldots\mathtt{z}I_{k+1}\mathtt{z}I_k\mathtt{z}I_{k-1}\mathtt{z}\ldots\mathtt{z}I_1\mathtt{zoi}\ldots\mathtt{izz},$$

where n is the number of states of P'_2, where the size of the *shuttle* $\mathtt{oi}\ldots\mathtt{iz}$ is equal to the longest size of the I_k's minus 5, and where, for all $a \in \Sigma_\mathtt{u}$ and $i \in 1..n$,

$$I_{\pi(i,a)} = \begin{cases} \overline{[q_i, abs, q_j]}, \text{ if there exists } b, s, j \text{ with } [q_i, abs, q_j] \in P'_2, \\ \mathtt{oi}, \qquad \text{otherwise,} \end{cases}$$

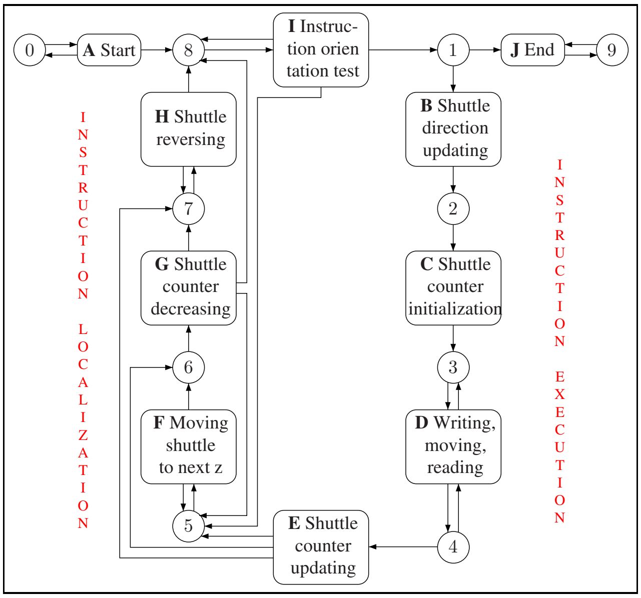
with,

$$\overline{[q_i, a, b, s, q_j]} = \begin{cases} \mathtt{i}a_m \ldots a_2\mathtt{o}, \text{ if } \pi(j) - h(i,a) > 0, \\ \mathtt{o}a_2 \ldots a_m\mathtt{i}, \text{ if } \pi(j) - h(i,a) < 0, \end{cases}$$

with a_2a_3 equal to $\mathtt{io}$, $\mathtt{oi}$, $\mathtt{ii}$, depending whether b equals $\mathtt{u}$, $\mathtt{o}$, $\mathtt{i}$, $\mathtt{z}$, with $a_4 = \mathtt{o}$ or $a_4 = \mathtt{i}$ depending whether $s = +$ or $s = -$ and with $\mathtt{i}a_m \ldots a_5$ a binary number ($\mathtt{o}$ for 0 and $\mathtt{i}$ for 1) whose value is equal to $|\pi(j) - \pi(i,a)| + \frac{3}{2}$.

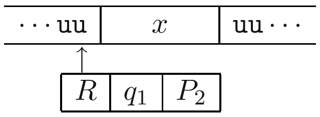
5.3 Operation of the Universal Pair $(U_2, code_2)$

As already mentioned, the program U_2 has 54 states, $q_1, \ldots, q_{54}$, and 184 instructions. These instructions are divided in 10 modules $A, B, C, \ldots, J$ organized as follows:

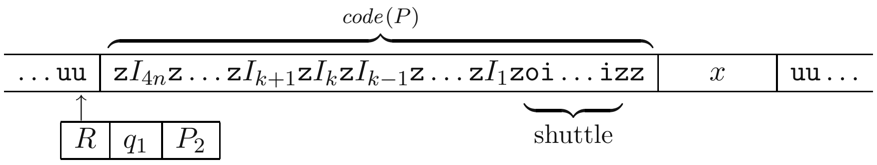

0
A Start
8
I Instruction orientation test
1
J End
9
H Shuttle reversing
B Shuttle direction updating
INSTRUCTION LOCALIZATION
7
2
INSTRUCTION EXECUTION
G Shuttle counter decreasing
C Shuttle counter initialization
6
3
F Moving shuttle to next z
D Writing, moving, reading
5
E Shuttle counter updating
4

The numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 denote respectively the states q_1, q_{24}, q_{35}, q_{43}, q_{49}, q_{15}, q_{13}, q_7, q_{10}, q_{23}. The complete program U_2 is given in the appendix as a graph whose vertices are the states and the edges the instructions of U_2: each instruction $[q_i, abs, q_j]$ is represented by an arrow, labeled abs, going from q_i to q_j. Note that the vertices a, b, c and 7 have two occurrences which must be merged.

Initial Configurations Initially the machines executing P_2 is in the configuration

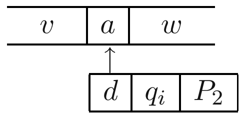


and the machine executing u_2 is in the *correponding initial configuration*

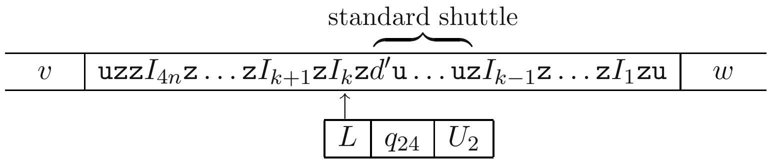

$code(P)$
…uu
$zI_{4n}z\ldots zI_{k+1}zI_kzI_{k-1}z\ldots zI_1zoi\ldots izz$
x
uu…
R
q_1
P_2
shuttle

While the machine executing P_2 performs no transitions, the machine executing U_2 performs a sequence of initial transitions, always the same, involving the instructions of module A and some instructions already there in the modules I, H, G, F. Then the machines executing P_2 and U_2 end up respectively in the following current configurations with $k = 1$:

Current Configurations While the machine excuting P_2 is in the current configuration

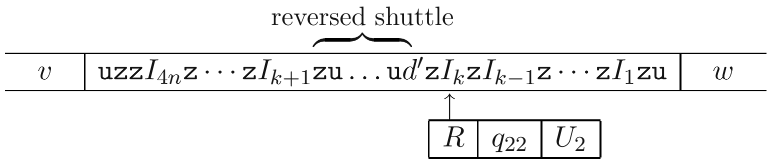


the machine executing U_2 is in the *corresponding current configuration*


standard shuttle
v
$\mathtt{uzz}I_{4n}\mathtt{z}\ldots\mathtt{z}I_{k+1}\mathtt{z}I_k\mathtt{z}d'\mathtt{u}\ldots\mathtt{uz}I_{k-1}\mathtt{z}\ldots\mathtt{z}I_1\mathtt{zu}$
w
L
q_{24}
U_2

(5)

or


reversed shuttle
v
$\mathtt{uzz}I_{4n}\mathtt{z}\cdots\mathtt{z}I_{k+1}\mathtt{zu}\ldots\mathtt{u}d'\mathtt{z}I_k\mathtt{z}I_{k-1}\mathtt{z}\cdots\mathtt{z}I_1\mathtt{zu}$
w
R
q_{22}
U_2

(6)

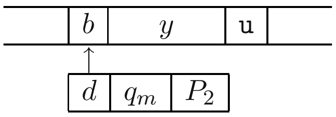
depending whether I_k, with $k = \pi(i, a)$, is in the *standard* form $\mathtt{i}a_m \ldots a_2\mathtt{o}$ or in the *reversed* form $\mathtt{i}a_m \ldots a_2\mathtt{o}$. The read-write points to a_3 or to the `z` which follows I_k when I_k is the empty instruction `oi`. Depending whether d is equal to L or R, the symbol d' is equal to `u` or `o`, if I_k is standard, and to `o` or `u`, if I_k is reversed.

While the current configuration of P_2 is not final, P_2 performs one transition for reaching the next current configuration and U_2 performs a sequence of transitions for reaching the next corresponding current configuration. More precisely, using the information contained in I_k, the program U_2

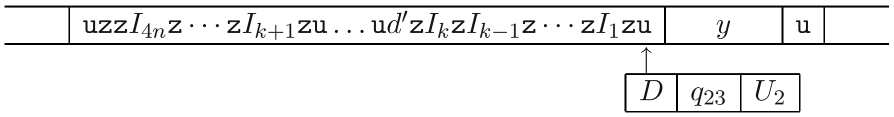
- updates the internal direction contained in the shuttle (module B),
- transfers in the shuttle the binary number serving as basis for computing the number of instructions to be jumped toward the left or the right, depending on whether the shuttle is standard or reversed (module C),
- simulates the writing of a symbol, the read-write head move, and then the reading of a new symbol (module D),
- taking into account the read symbol, updates the binary number contained in the shuttle in order to obtain the right number of instructions to be jumped by the shuttle for reaching the next instruction to be executed (module E),
- moves the shuttle and eventually reverses it, for correctly positioning it alongside the next instruction to be executed (modules F, G, H, I).

When the current configuration of P_2 becomes final, the corresponding current configuration of U_2 is of the form (6) with I_k equal to the empty instruction `oi`. Then U_2 performs a sequence of transitions (module J) for reaching the final corresponding configurations. The machines executing P_2 and U_2 end up respectively in the following final configurations:

Final Configurations While the machine executing P_2 terminates in the final configuration



the machine executing U_2 terminates in the *corresponding final configuration*



$\texttt{uzz}I_{4n}\texttt{z}\cdots\texttt{z}I_{k+1}\texttt{zu}\ldots\texttt{u}d'\texttt{z}I_k\texttt{z}I_{k-1}\texttt{z}\cdots\texttt{z}I_1\texttt{zu}$
y
u
D
q_{23}
U_2

with $I_k = \texttt{oi}$, $k = \pi(m, b)$ and d' equal to o or u, depending whether d equals L or R.

6 Complexity and Introspection Coefficient of Our Two Pairs

6.1 General Complexity

Let P_2 be any program for $\mathcal{M}_2$. From the way the coding function $code_2$ is defined, for $\ell = 2$,

$$\boxed{|code_\ell(P_\ell)| = \mathcal{O}(n \log n)},$$

where n is the number of states of $clean(P_\ell)$. This result also holds for $\ell = 1$, with P_1 being any program for $\mathcal{M}_1$, the classical Turing mchine with same alphabet as $\mathcal{M}_2$. This is due to the fact that $code_1(P_1) = code_2(f_1(P_1))$ and that the number of instructions of $clean(f_1(P_1))$ is at most equal to twice the number of instructions of P_1.

Returning to the way U_2 operates at section 5.3, we conclude that if P_2 performs h transitions then there exists positive integers k_i indepent of h or n such that U_2 performs at most:

– $k_1 \log^2 n$ transitions for reaching the first configuration corresponding to the initial configuration of P_2,
– $hk_1 \log^2 n$ transitions for transferring information from an instruction I_i to the adjacent shuttle,
– $hk_2 n \log n$ transitions for simulating the writing of a symbol, the move of the head and the reading of a symbol,
– $hk_3 n \log^2 n$ transitions for moving the shuttle,
– $hk_4 n \log^2 n$ transitions for reaching a final configuration from a configuration corresponding to a final configuration of P_2,

that is all together, at most $hk_5 n \log^2 n$ transitions.

Thus for $\ell = 2$, there exists a positive real number k, independent of $x \in \Sigma^\star$, such that

$$\boxed{\lambda_\ell(P_\ell, x) \leq n \log^2 n}$$

If instead of measuring the complexity in terms of n we do it in terms of $m = |code_\ell(P_\ell)|$, we conclude that there exists a positive real number k, independent of $x \in \Sigma^\star$, such that

$$\boxed{\lambda_\ell(P_\ell, x) \leq m \log m}$$

These two results also hold also for $\ell = 1$, with P_1 being any program for $\mathcal{M}_1$.

6.2 Complexity on Examples

On particular examples we have obtained the following complexity results for the pairs $(U_\ell, code_\ell)$, with $\ell = 1$ and $\ell = 2$,

x	$cost(P_\ell, x)$	$cost(U_\ell, code_\ell(P_\ell)\cdot x)$	$cost(U_\ell, code_\ell(U_\ell)\cdot code_\ell(P_\ell)\cdot x)$	$\lambda_\ell(P_\ell, x)$	$\lambda_\ell(U_\ell, code_\ell(P_\ell)\cdot x)$
ε	2	5 927	22 974 203	2 963.50	3 876.19
`o`	6	13 335	51 436 123	2 222.50	3 857.23
`oi`	12	23 095	88 887 191	1 924.58	3 848.76
`oiz`	20	35 377	136 067 693	1 768.85	3 846.22
`oizo`	30	49 663	190 667 285	1 655.43	3 839.22

Here P_ℓ, with $P_1 = f_2(P_2)$, is a reversing program such that, for all $n \geq$, one gets $out(P_\ell,\ a_1a_2\ldots a_n) = a_n \ldots a_2a_1$, with the a_i's taken from $\{\mathtt{o}, \mathtt{i}, \mathtt{z}\}$. In both cases, $\ell = 1$ and $\ell = 2$, the program P_ℓ has 32 instructions and 9 states. We have $|code_\ell(P_\ell)| = 265$ and $|code_\ell(U_\ell)| = 1552$.

6.3 Introspection Coefficient

To satisfy Hypothesis 2, $code_2(U_2) = f_1(f_2(code_2(U_2)))$ and the labelling function μ is defined so that, for all transitions (c_1, c_2) and (c_1', c_2') of $\bigcup U_2$ the integers $\mu(c_1, c_2)$ and $\mu(c_1', c_2\prime)$ are equal if and only if all the conditions below are satisfied:

- the states of c_1 and c_1' are equal,
- the symbols pointed by the read-write heads in c_1 and c_1' are equal,
- the symbols pointed by the read-write heads in c_2 and c_2' are equal,
- the directions in c_2 and c_2' are the same.

The function Φ is defined, for all configuration of U_2, with $P_2 = U_2$ by,

$$\Phi(c) = \begin{cases} \text{current configuration corresponding to } c, \text{ if } c \text{ is not final for } P_2, \\ \text{final configuration corresponding to } c, \text{ if } c \text{ is final for } P_2. \end{cases}$$

After having computed the column vector B and the matrix A, using Theorem 2, we have verified that U_2 admits an introspect coefficient and computed its value: for $\ell = 2$ and all words x on Σ such that $cost(P,\ x) \neq \infty$,

$$\boxed{\lim_{n\to\infty} \frac{cost(U_\ell,\ code_\ell(U_\ell)^{n+1}\cdot x)}{cost(U_\ell,\ code(U_\ell)^{n}\cdot x)} = 3\,672.98}$$

This result also holds for $\ell = 1$.

7 Our Universal Program for the Indirect Addressing Arithmetic Machine

It is interesting to compare the complexities of our universal program for a Turing machine with the complexity of a universal program for the indirect addressing arithmetic machine with same alphabet $\Sigma = \{c_1, c_2, c_3\}$, with $c_1 = \mathtt{o}$, $c_2 = \mathtt{i}$ and $c_3 = \mathtt{z}$.

We have written such a universal program U_3 using 103 instructions. From the operation of our universal pair (U_3, code_3) we have been able to show that:

Property 6 *There exists a positive number k such that, for all programs P_3 and word x on Σ, with $\mathit{cost}(P_3, x) \neq \infty$,*

$$\boxed{\lambda_3(P_3, x) = \frac{\mathit{cost}(U_3,\ \mathit{code}_3(P_3)\cdot x)}{\mathit{cost}(P_3,\ x)} \leq 35 + k\frac{|\mathit{code}(P)|}{\mathit{cost}(P_3,\ x)}}$$

Thus the size of the program P_3 becomes irrelevant when a large number of transitions is performed. On particular examples we have obtained the following results:

x	$\mathit{cost}(P_3, x)$	$\mathit{cost}(U_3, \mathit{code}_3(P_3)\cdot x)$	$\mathit{cost}(U_3, \mathit{code}_3(U_3)\cdot \mathit{code}_3(P_3)\cdot x)$	$\lambda_3(P_3, x)$	$\lambda_3(U_3, \mathit{code}_3(P_3)\cdot x)$
ε	12	2 372	72 110	197,66	30,40
`o`	16	2 473	74 758	154,56	30,23
`oi`	31	2 860	84 916	92,26	29,69
`oiz`	35	2 961	87 564	84,60	29,57
`oizo`	50	3 348	97 722	66,96	29,19

where P_3 is a reversing program of 21 instructions such that, for all $n \geq 0$ one obtains $\mathit{out}(P,\ a_1a_2\ldots a_n) = a_n \ldots a_2a_1$, with the a_i's taken from $\{\mathtt{o}, \mathtt{i}, \mathtt{z}\}$. Additionally, $|\mathit{code}_3(P_3)| = 216$ and $|\mathit{code}_3(U_3)| = 1042$.

The introspection coefficient obtained is:

$$\boxed{\lim_{n\to\infty} \frac{\mathit{cost}(U_3,\ \mathit{code}(U_3)^{n+1}\cdot x)}{\mathit{cost}(U_3,\ \mathit{code}(U_3)^{n}\cdot x)} = 26.27}$$

8 Conclusion

Unless one “cheats”, it is difficult to improve the introspection coefficient of our universal Turing machine which took us considerable development effort. Suppose, which is the case, that we have at our disposal a first universal pair (U, code) for a Turing machine.

A first way of cheating consists of constructing the pair (U, code') from the universal pair (U, code), with

$$\mathit{code}'(P) = \begin{cases} \varepsilon, & \text{if } P = U, \\ \mathit{code}(P), & \text{if } P \neq U. \end{cases}$$

Then we have
$$\frac{cost(U',\, code(U')^{n+1}\cdot x)}{cost(U',\, code(U')^{n}\cdot x)} = \frac{cost(U',x)}{cost(U',x)} = 1$$
and $(U, code')$ is a universal pair with an introspection coefficient equal to 1.

There is a second more sophisticated way of cheating, without modifying the coding function *code*. Starting from the universal program U we construct a program U', which, after having erased as many times as possible a given word z occurring as prefix of the input, behave as U on the remaining input. According to the recursion theorem [2,4], it is possible to take z equal to $code(U')$ and thus to obtain a universal program U' such that, for all $y \in \Sigma^\star$ having not $code(U)'$ as prefix,
$$cost(U',\; code(U')^n \cdot y) = nk_1 + k_2(y),$$
where k_1 and $k_2(y)$ are positive integers, with k_1 being independent of y. Then we have
$$\frac{cost(U',\, code(U')^{n+1}\cdot y)}{cost(U',\, code(U')^{n}\cdot y)} = \frac{cost(U,x)+(n+1)k_1+k_2(y)}{cost(U,x)+nk_1+k_2(y)} =$$
$$1 + \frac{k_1}{cost(U,x)+k_2(y)+nk_1}.$$
By letting n tend toward infinity we obtain an introspection coefficient equal to 1 for the pair $(U', code)$.

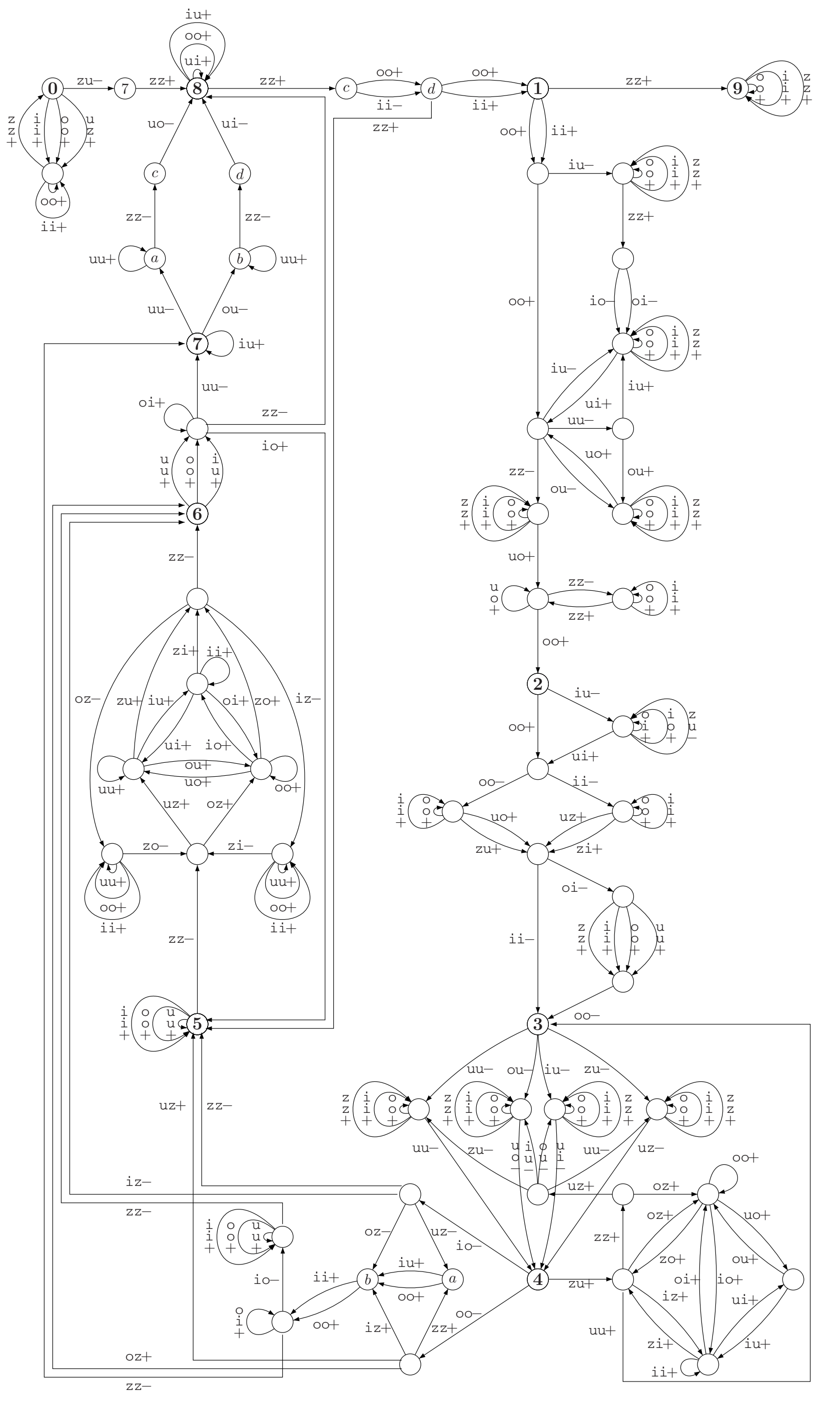
Unfortunately our introspection coefficient definition, page 25, does not disallow these two kinds of cheating. By imposing Hypothesis 2, the first way of cheating is still possible and the second one can be prevented by imposing that the function φ never produces the empty sequence. But this last restriction seems to be *ad hoc*.

What one really would like to prevent is that the function *code* or the program U "behaves differently" on the program P, depending whether P is or is not equal to U. It is an open problem to express this restriction in the definition of the introspection coefficient.

References

1. Marvin Minsky, *Computations: Finite and Infinite Machines,* Prentice-Hall, 1967.
2. Hartley Rogers, *Theory of Recursive Functions and Effective Computability,* McGraw-Hill, 1967, also MIT Press, fifth printing, 2002.
3. Yurii Rogozin, Small universal Turing machines, *Theoretical Computer Science,* Volume 168, number 2, november 1996.
4. Michael Sipser, *Introduction to the Theory of Computation,* PWS Publishing Company, 1997.

Appendix: Graph of the Universal Program U_2



Finite Sets of Words and Computing[*]
(A Survey)

Juhani Karhumäki

Department of Mathematics and
Turku Centre for Computer Science
University of Turku
FIN-20014 Turku, Finland
karhumak@cs.utu.fi

Abstract. We discuss about two recent undecidability results in formal language theory. The corresponding problems are very simply formulated questions on finite sets of words. In particular, these results underline how finite sets of words can be used to perform powerful computations.

1 Introduction

In the past few years two important problems on finite sets of words have been solved. First in [KL03b] it was shown that the equivalence problem for finite substitutions on the language ab^*c is undecidable. Then, very recently, M. Kunc, see [Ku05], solved the 30-year-old Conway's Problem even in a very strong form, namely by showing that the maximal set commuting with a given rational set needs not be rational, nor even recursively enumerable.

The solutions have at least two common features. First of all they show an undecidability in a very simple set-up for finite languages. Second, both of the solutions are a bit surprising – the answers were, at least at the very beginning, expected to be opposite.

A common bottomline of both of these solutions is that they show how finite sets can be used to simulate powerful computing processes. This is the point we want to make in this presentation. Consequently, our goal is to recall major results related to above mentioned problems, as well as to discuss the computational processes used, not in technical but in informal level. So no complete proofs are presented here.

After preliminaries in Section 2 we consider so-called morphic mappings, that is mappings which are compositions of morphisms and inverse morphisms. We recall their close connections to finite transductions which are mappings realizable by finite state machines. In Section 4 we move to our first important problem: the undecidability of the equivalence problem of finite substitutions on the language ab^*c. Here the computational power of finite substitutions is illustrated. In Section 5 we ask why the above result is important, and try to

[*] Supported by the Academy of Finland under the grant 44087

answer this mainly via applications of the result. In Section 6 we conclude with the recent solution of Conway's Problem, as well as state some related results. These together with results of Section 4 emphasize a drastic difference between the "nondeterminism" and the "unambiguity" in certain problems.

2 Preliminaries

We assume that the reader is familiar with the basics of formal language theory, see e.g. [Sa73],[HU79] or [Be79]. The following lines are mainly to fix the used terminology.

We denote a finite *alphabet* by A, the free monoid (resp. semigroup) it generates by A^* (resp. A^+). Elements of A^* are called *words*, and subsets of A^* are called *languages*. A *morphism* from A^* into B^* is a mapping h satisfying $h(uv) = h(u)h(v)$ for all $u, v \in A^*$. In particular, the image of the *empty word* 1 goes to the empty word under h, i.e. $h(1) = 1$. The *inverse* of a morphism h^{-1} is a partial mapping from B^* into 2^{A^*}, that is into the monoid of languages over A. By a *morphic* mapping we mean any composition of morphisms and inverse morphisms. We denote the submonoid of 2^{B^*} consisting of all finite languages by $\mathrm{Fin}(B)$.

A *finite substitution* σ is a morphism from A^* into $\mathrm{Fin}(B)$. Hence to define it it is enough to give the values $h(a)$ for all $a \in A$. A finite substitution σ can be decomposed as $\sigma = h \circ c^{-1}$, where c is a *length preserving* morphism often referred to as a *coding*. In particular, a finite substitution is a special case of morphic mappings of the form $h_1 \circ h_2^{-1}$, where h_1 and h_2 are morphisms.

We denote by $\mathcal{H}$, $\mathcal{H}^{-1}$ and $\mathcal{FS}$ the families of morphisms, inverse morphisms and finite substitutions. Similarly, for example $\mathcal{H} \circ \mathcal{H}^{-1}$ denotes the family of morphic mappings of the form $h_2 \circ h_1^{-1}$ where h_1 and h_2 are morphisms. Finally, by $\mathcal{H}^*$ we denote the family of all morphic mappings.

As is well known a language $L \subseteq A^*$ is *rational* if it is accepted by a *finite automaton* $\mathcal{A}$, denoted as $L = L(\mathcal{A})$. Similarly, a mapping $\tau : A^* \to 2^{B^*}$ is *rational* if it is computed by a *finite transducer*, e.g. a finite automaton with outputs. Viewing τ as a many valued mapping we often write $\tau : A^* \to B^*$. We do not present the formal definition of above notions. Instead we recall that finite automata are described by transitions

$$p \xrightarrow{a} q$$

and finite transducers by transitions

$$p \xrightarrow{a,\alpha} q.$$

Here p and q denote the states, a the input symbol and α the output word associated to this transition. If the cardinality of the input alphabet is one the automaton (the transducer) is *unary*. They are *deterministic* if q (resp. α and q) is unique for each pair (a, p). Finally, a transducer is called *input deterministic* if q (but not necessarily α) is unique for each pair (a, p). (Actually in the two last definitions we assume also the unique initial state.) We call two automata (resp. finite transducers) *equivalent* if they define the same language (resp. transduction).

Two basic results are

Theorem 1. *(Folklore) (i) The equivalence problem for deterministic finite transducers is decidable.*

(Griffiths, 1969) (ii) The equivalence problem for finite transducers is undecidable.

A remarkable extension of Theorem 1 (ii) is as follows:

Theorem 2. *(Ibarra, 1978; Lisovik, 1979) The equivalence problem for unary finite transducers is undecidable.*

Note that unary above means with respect to the input (or output but not both) alphabets. If both the alphabets are unary the problem is trivially decidable.

A variant of finite transducers is a so-called *defence system.* It is a finite state machine where transitions are of the form

$$p \xrightarrow{a,n} q,$$

where p, q and a are as above, but n is an integer, so-called *weight.* The machine contains just one initial state, and all states are final. It is allowed to be nondeterministic. A *weight* of a computation is the sum of the weights on the corresponding path. We say that a computation is *defending* if its weight is zero. Finally, a defence system is called *reliable* if every input word has a defending computation.

Now, Theorem 1 (ii) can be modified to (by using PCP):

Theorem 3. *(Lisovik, 1990) It is undecidable whether a given defence system is reliable.*

Some further notions are defined later when needed.

3 Morphic Mappings

In this section we review a few results on morphic mappings, mainly as a background material for the next section. We start by two representation results.

Theorem 4. *([KL83], [LL83]) Each rational transduction $\tau : A^* \to B^*$ allows a decomposition*

$$\tau = h_4 \circ h_3^{-1} \circ h_2 \circ h_1^{-1} \circ m,$$

where each h_i is a morphism and m is a marking adding a right endmarker to a word.

Idea of the proof. Instead of proving the result here we present an illustration which explains how the morphisms and inverse morphisms can be used to simulate computations of a finite transducer T realizing τ. So let us consider a computation

$$i \xrightarrow{a_0,\ldots,a_t,\alpha_0,\ldots,\alpha_t} t,$$

where a_i's are letters, α_i's are words and $p_i \xrightarrow{a_i,\alpha_i} p_{i+1}$ corresponds a step in the computation. The illustration is as in figure 1.

Here we first mark the input (for technical reasons not explained here), then guess a sequence of transitions and transform it to a single word. Next step is crucial: We select those sequences which correspond computations. Here an essential point is that in the previous coding the inputs are separated by a constant number of zeros (that is $j = i'$ meaning that the states q and p' coincide). Finally, we project the output from the computation. Note that if T never outputs the empty word all morphisms above are nonerasing. □

The above technique, and its variants, allowes to prove many versions and sharpening of Theorem 4. For example, we have a characterization of morphic mappings.

Theorem 5. *(Latteux, Turakainen, 1987) Each morphic mapping H can be written in the form*

$$H = h_4^{-1} \circ h_3 \circ h_2^{-1} \circ h_1.$$

When combining these results with Theorem 1 we obtain:

Corollary 1. *The equivalence of two morphic mappings is undecidable.*

In order to sharpen this result, and in order to search for a borderline between the decidability and the undecidability, we recall the problem area defined by Culik II and Salomaa, see [CS78]. Let $\mathcal{L}$ be a family of languages, like those of rational and context-free languages $\mathcal{R}$ and $\mathcal{CF}$, respectively. Further, let f and g be mappings on a language $L \in \mathcal{L}$. We say that f and g *are equivalent* on L, in symbols $f \overset{L}{\equiv} g$, if

$$f(w) = g(w) \quad \text{for all} \quad w \in L.$$

The equivalence problem of $\mathcal{F}$ on $\mathcal{L}$ asks to decide, for two given mappings $f, g \in \mathcal{F}$ and a language $L \in \mathcal{L}$, whether h and g are equivalent on L.

A few simple examples are:

Example 1. The equivalence of morphisms on rational languages is decidable – due to the pumping property of $\mathcal{R}$ and a simple combinatorial lemma on words.

Example 2. The equivalence of morphisms on context-free languages is decidable as well, see [CS78], although clearly more complicated than the problem of Example 1.

Now, we can state a strict borderline between the decidability and the undecidability in the above set-up.

$$\begin{tikzcd}
a_0a_1\cdots\ \cdots aa'\cdots\ \cdots a_t \arrow[d, "\cdot m\qquad\qquad \text{mark}"] \\
a_0a_1\cdots\ \cdots aa'\cdots\ \cdots a_t\# \arrow[d, "h_1^{-1}\qquad\qquad \text{guess}"] \\
\cdots(p,a,q)(p',a',q')\cdots \arrow[d, "h_2\qquad\qquad \text{transform}"] \\
\cdots 0^i a\underbrace{0^{m-j}0^{i'}}_{m\text{ zeros}}a'0^{m-j'}\cdots \arrow[d, "h_3^{-1}\qquad\qquad \text{select}"] \\
(p,a,q)(q,a'q') \arrow[d, "h_4\qquad\qquad \text{project}"] \\
\alpha_0\alpha_1\cdots\ \cdots\alpha\alpha'\cdots\ \cdots\alpha_t
\end{tikzcd}$$

Fig. 1.

Theorem 6. *[KK85] The equivalence problem of $\mathcal{H}^{-1}\circ\mathcal{H}$ on $\mathcal{R}$ is decidable while that of $\mathcal{H}\circ\mathcal{H}^{-1}$ on $\mathcal{R}$ is undecidable.*

What remained open here is a special case of the latter result. Indeed, finite substitutions are special cases of the mappings of the form $\mathcal{H}\circ\mathcal{H}^{-1}$, in fact they are in $\mathcal{H}\circ\mathcal{C}^{-1}$, where $\mathcal{C}$ denotes the family of codings (so that their inverses are *renamings* of the letters).

This problem was formulated in [CuK83], and more explicitly in [K85], and actually expected to be decidable. It, however, turned out to be very challenging. This was noticed very soon, e.g. by a result of J. Lawrence, see [La86], which shows that even the rational language ab^*c does not possess a *finite test set*, that is a finite subset such that to test whether two finite substitutions are equivalent on L it suffices to check this on F.

Consequently, there remained the following two problems.

Problem 1. Is the equivalence problem of finite substitutions on rational languages decidable?

Problem 2. Is the equivalence problem of finite substitutions on the regular language ab^*c decidable?

These are the problems of the next section.

4 Finite Substitutions

This section is devoted to the first fundamental result of this survey. We consider Problems 1 and 2 introduced in the previous section. The first breakthrough result was proved by L. Lisovik.

Theorem 7. *(Lisovik, 1997) It is undecidable whether two finite substitutions are equivalent on regular languages. In fact, the regular language can be chosen to be $a\{b,c\}^*d$.*

Note that Theorem 7 was reproved (and explained) in [HH99]. As noted already in [CuK86], Theorem 7 has the following corollary – essentially due to the fact that the set of accepting computations of a finite automaton forms a rational language over the set of its transitions.

Corollary 2. *The equivalence problem of input deterministic finite transducers is undecidable.*

The next step was to sharpen the construction of the proof of Theorem 7 in order to replace the language $L_1 = a\{b,c\}^*d$ by the language $L_2 = ab^*c$. Note that these two languages are in many aspects very different: the first one is unbounded as well as essentially over a binary alphabet, while the second one is bounded and essentially over the unary alphabet. What was proved in [KL03a] was that L_1 can be replaced by L_2 if at the same time the equality is relaxed to the *inclusion*, that is the question

$$''\text{Is } \ \varphi(ab^nc) \subseteq \psi(ab^nc) \ \text{ for all } \ n \geq 0?''$$

is asked for given two finite substitutions φ and ψ.

Theorem 8. *([KL03a]; Turakainen, 1988) The inclusion problem of two finite substitutions on ab^*c is undecidable.*

Actually, this was proved also in [Tu88]. Finally, in [KL03b] a further extension was achieved to obtain

Theorem 9. *([KL03b]) It is undecidable whether two finite substitutions are equivalent on the language ab^*c.*

Idea of the proof. Without going into a formal proof (which is rather long) we want to illustrate the main construction used in it. It resembles the construction shown in Theorem 4, however, now we need much more involved codings. The result is reduced to Theorem 3, that is defending computations are simulated by finite substitutions on ab^*c.

First we note that the defence system can be assumed so that the only weights are −1, 0 and 1. The illustration is as shown in figure 2.

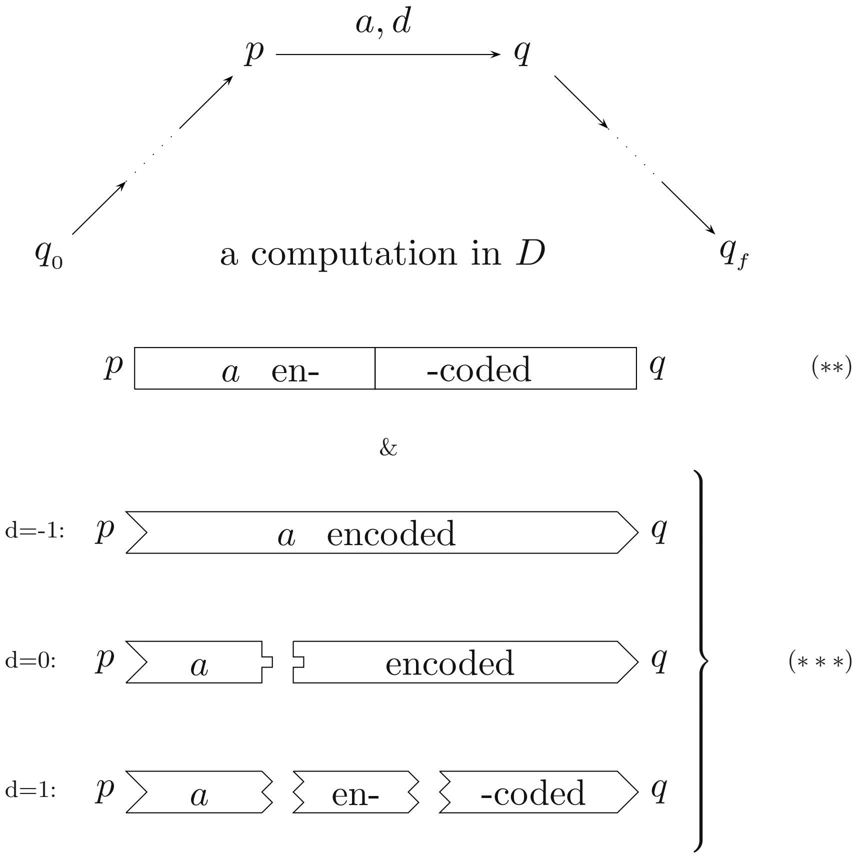


Fig. 2.

So transitions are encoded, as in the proof of Theorem 4. Now this is done in two ways: First by encoding the states and the input symbol using always two blocks of words (case (**)), and *also* in another way, where the weights are taken into account (case (***)). In this latter encoding the number of blocks is either 1, 2 or 3 depending on whether the weight of the transition is -1, 0 or 1. The above blocks are images of the letter b under the both finite substitutions φ and ψ. Actually, the image of b consists always of two "consecutive" blocks under both of the finite substitutions.

Without going into details, for which we refer to [KL03b], we recall that the reliability of the defence system D considered is related to the equivalence of φ and ψ on ab^*c as follows. Defending computations are those where the total weight is 0. A computation of length n according to encodings of $(**)$ produces $2n$ blocks of words. Equally many are obtained according to $(***)$ using the same number n of b's by taking the first alternative as many times as the third. This is how the counting of b's is related to the safety of computations. There are several technical matters not explained here. One thing, however, should be emphasized: As we said, both φ and ψ are equally defined on b. In order to make use of the above idea of having the second computation (which checks the safety) we must be able to unbalance the computations at the beginning, that is on a. This, indeed, can be done by allowing $\varphi(a)$ to have one extra value compared to those of $\psi(a)$. With the exception of this one value φ and ψ are identically defined.

It follows that D is reliable if and only if φ and ψ are equivalent on the language ab^*c. □

5 Applications

In this section we raise a question why the result of the previous section is interesting. Obvious answers are that it solves a simply formulated longstanding open problem in an interesting research topic. And even more importantly the solution is not what was expected at the beginning.

However, we believe, that there are even more important answers. Namely, the result has a few fundamental consequences. First, it extends the undecidability of the equivalence problem for finite transducers to amazingly simple machines. Let T be a finite transducer of the type depicted as in figure 3.

$(*)$

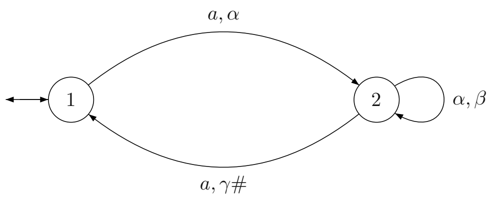


Fig. 3.

So in T

- there are just two states, and
- the input alphabet is unary.

We have:

Theorem 10. *([KL03b]) The equivalence problem of finite transducers of type $(*)$ is undecidable.*

The result follows directly from Theorem 9 by identifying the triples (α, β, γ) with the values $(\varphi(a), \varphi(b), \varphi(c))$ of a finite substitution φ. Different input symbols are not needed since they can be encoded to states of the transducer. However, the marker $\#$ is essential.

The other application is to systems of equations over finite sets of words. We need some terminology. Let X be a finite set of unknowns and A a finite alphabet. An *equation* over A^* with X as the set of unknowns is a pair $(u, v) \in (X \cup A)^+ \times (X \cup A)^+$, usually written as $u = v$. A *solution* of an equation $e : u = v$ is a morphism $h : (X \cup A)^* \to A^*$ identifying u and v and being the identity on letters of A, that is

$$h(u) = h(v) \quad \text{and} \quad h(a) = a \quad \text{for all} \quad a \in A.$$

Further a system of equations is a set of equations. We call a system *rational*, if there exists a finite transducer translating the left hand sides of the equations to the corresponding right hand sides. For example, the system $S = \{aby^n z = w^{2n} | n \geq 0\}$ is rational. The solutions of systems of equations are defined in the natural way.

A fundamental problem is the *satisfiability problem* for equations (or systems of equations). It asks to decide whether a given equation (or system of equations) possesses a solution. Note that in order to avoid trivialities the above equations are with constants.

Everything above was defined with respect to word monoids A^*, but extends in a natural way to all monoids and semigroups. In particular, we can talk about the satisfiability problem for finite sets of words, i.e. in the monoid $\mathrm{Fin}(A)$. Of course, in this case the constants in the equations are finite languages.

In the word case we have the following fundamental results.

Theorem 11. *(Makanin, 1976) (i) The satisfiability problem for word equations is decidable.*

(ii) The satisfiability problem for finite sets of word equations is decidable.

(iii) The satisfiability problem for rational sets of word equations is decidable.

A breakthrough result here was that of S. Makanin. Indeed, even before that it was known that (i) and (ii) are equivalent, and the equivalence of (ii) and (iii) is not very hard either, see [CuK83]. The second fundamental achievement on this area was a recent paper by W. Plandowski, see [Pl04], which not only gave a new solution for problem (i), but also showed that it is in $PSPACE$.

When we move from words to finite sets of words the situation changes drastically: only in (iii) the decidability status is known, and even then the problem is undecidable. This is the second application of Theorem 10.

Theorem 12. *([KL03b]) The satisfiability problem of rational systems of equations in* $\mathrm{Fin}(A)$ *is undecidable.*

In fact, it is not only that Theorem 12 holds, but also it is undecidable whether a given candidate is a solution of a given rational system of equations over $\mathrm{Fin}(A)$. In other words, not only the "emptiness" problem but also the "membership" problem is undecidable here.

So what remains is

Problem 3. Is the satisfiability problem for a single equation decidable in $\mathrm{Fin}(A)$?

Although we do not know whether (iii) and (i) (or even (ii) and (i)) are equivalent in the monoid of finite languages, Theorem 12 is an evidence that Problem 3 is likely to be hard – at least to prove the decidability. Another evidence is that even a single unknown variant is open:

Problem 4. Is it decidable whether, for two given finite sets A and B, the equation

$$Az = zB$$

has a solution?

In other words, Problem 4 asks whether two finite sets are *conjugates.*

6 Conway's Problem

In this section we consider another fundamental problem on finite (or rational) sets of words. Namely, we consider the commutation equation

$$xy = yx. \tag{1}$$

As is well known, see e.g. [Lo83] or [CK97], in the word monoid (1) is equivalent to the fact that x and y are powers of a common word. In the monoid of finite languages, that is in $\mathrm{Fin}(A)$, a characterization is not likely to be achievable. Even much simple problems are hard.

Conway's Problem is one of the challenging problems. It was presented by J. Conway in 1971, see [Co71], asking whether the maximal set commuting with a given finite set is rational. Actually, he asked the question for rational instead of finite sets. For a given $X \subseteq A^*$ we denote the above maximal set by $\mathcal{C}(X)$ and call it the *centralizer* of X.

It is straightforward to see that $\mathcal{C}(X)$ always exists – it is the union of all sets commuting with X. It is also a monoid or a semigroup (depending on whether the empty word is the considerations). Finally, $\mathcal{C}(X)$ is a superset of X^* and a subset of all prefixes in X^*, i.e.

$$X^* \subseteq \mathcal{C}(X) \subseteq \mathrm{Pref}(X^*).$$

It turned out that not only the Conway's Problem is hard, but even to show that the centralizer is recursive or recursively enumerable was a challence. Obvious ways to attack this were missing. A reason for this can be illustrated as follows. Assume that $z \in \mathcal{C}(X)$. Then, for any $x \in X$, there must be $x' \in X$ such that $z' = x^{-1}zx'$ is in $\mathcal{C}(X)$ (and the same to the left). This can be depicted as in figure 4.

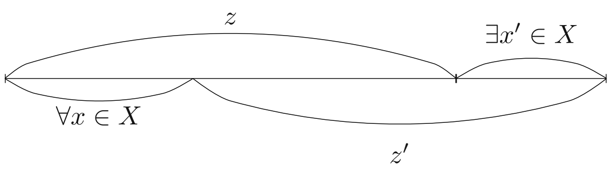


Fig. 4.

The procedure can be continued, but how to conclude that actually z' (and hence also z) is in $\mathcal{C}(X)$. This is a real difficulty!

A simplified related question is arised in the next example.

Example 3. Let $X \subseteq A^+$ be a finite set and $\omega \in A^+$ a word. We define a rewriting rule by the condition

$$u \Rightarrow v \quad \text{iff} \quad \exists x, x' \in X : \; v = x^{-1}ux'. \tag{2}$$

As usual let $\Rightarrow^*$ be the transitive and reflexive closure of $\Rightarrow$, and

$$OCC(\omega) = \{w \in A^* | \omega \Rightarrow^* w\}.$$

We can say that $OCC(w)$ is the (one-way) *orbit closure under commutation*. Although the above two problems resemble each other there is a crucial difference: In Conway's Problem we have the quantification "$\forall x \in X$" while in OCC-problem we do not have this. Consequently, in the latter case if the rule can be applied result is known to be in the language.

It is not difficult to see that $OCC(\omega)$ is always context-free (in fact, even one counter language), but to show that it is rational is more demanding. We can modify (2) in a natural way to 2-way rewriting by setting "$v = x^{-1}ux'$ or $v = x'ux^{-1}$" instead of "$v = x^{-1}ux'$". Amazingly, we do not know how the family of languages thus obtained is related to Chomsky hierarchy. □

Now, let us return to Conway's Problem. We present here only a few basic results on it, for a more complete survey we refer to [KP04]. In order to formulate the first results we recall that a *prefix* set is a set where no word is a prefix of another, and that each prefix set L possesses the unique *minimal root* $\rho(X)$ (due to the fact that the monoid of prefix sets is free).

Theorem 13. *(Ratoandromanana, 1989) (i) The centralizer of a prefix set L is $(\rho(L))^*$.*

[KP02] (ii) The centralizer of a three-element set $L = \{u, v, w\}$ is L^*.

Interestingly, in both of these cases we can characterize all the sets commuting with L:

$$XL = LX \Leftrightarrow \exists I \in \mathbf{N} : \; X = \bigcup_{i \in I} \rho(L)^i. \tag{3}$$

For 4-element sets such a characterization does not hold any more, see [CKO02]. Simple, but not trivial, proofs of Theorem 13 can be found in [KLP05] and [KLP03]. Actually, the characterization (3) is a nice exercise even for 2-element sets, see e.g. [BK04].

Many approaches have been developed to attack Conway's Problem, see [Pe02] and [KP04]. However, they were successfull only in very restricted cases. This was recently explained by M. Kunc in his breakthrough result solving the general conjecture, and even in the very strong form:

Theorem 14. *(Kunc, 2005) The centralizer of a finite set need not be recursively enumerable.*

The proof of Theorem 14 is technically pretty complicated, but very much in the spirit of this paper. It shows how finite sets of words can be used to simulate powerful computations, in this case tag-systems cf. [Mi67].

Another recent result is a solution of Conway's Problem for all codes, for definitions see [BP85].

Theorem 15. *[KLP05] The centralizer of a finite (or even rational) code is rational.*

A drastic difference between Theorems 14 and 15 is interesting. It is, we believe, explained by the fact that for a code X the product

$$X \cdot \mathcal{C}(X)$$

is unambiguous, and hence so is the product XY for any Y commuting with X. At least this unambiguity plays a central role in the proof. Note also that despite of Theorem 15, we do not know whether the characterization (3) holds for all codes.

7 Concluding Remarks

We have discussed two problems on finite sets of words. These problems are very simply formulated and natural ones. However, they both were open for decades, until very recently both were solved. Although the problems themselves are not related, and neither are their solutions, it turned out that both of the solutions were based on an interesting property. Namely, to the capability of finite sets of words to carry out powerful computations. This is the point we wanted to make in this presentation.

There is also another aspect which should be emphasized. The power of finite sets in our problems seems to be based on ambiguity. If we take unambiguous instances of the problems they will be essentially simpler. Indeed, for codes, that is for sets where the product $X \cdot \mathcal{C}(X)$ is ambiguous, the centralizer becomes rational. Similarly, the equivalence problem of prefix substitutions, that is substitutions for which images of letters are prefix sets, becomes decidable on regular languages, see [KL99].

Acknowledgement: The author is grateful to Dr. A. Okhotin for clarifying Example 3 and P. Salmela for his help in finalizing the manuscript.

References

[Be79] J. Berstel, *Transductions and Context-Free Languages*, Teubner, 1979.

[BK04] J. Berstel and J. Karhumäki, Combinatorics on Words – A Tutorial, in: G. Paun, G. Rozenberg and A. Salomaa (eds), Current Trends in Theoretical Computer Science, The Challences of the New Century, World Scientific, Singapore (2004), 415–476.

[BP85] J. Berstel and D. Perrin, *Theory of Codes*, Academic Press, New York (1985).

[CuK83] K. Culik II and J. Karhumäki, Systems of equations and Ehrenfeucht's conjecture, Discr. Math. 43, 1983, 139–153.

[CuK86] K. Culik II and J. Karhumäki, The equivalence problem of finite valued transducers (on $HDT0L$ languages) is decidable, Theoret. Comput. Sci. 47, 1986, 71–84.

[CK97] C Choffrut and J. Karhumäki, Combinatorics on Words, in: Rozenberg, G., Salomaa, A. (eds.), *Handbook of Formal Languages*, Vol. 1, Springer-Verlag (1997), 329–438.

[CKO02] C. Choffrut, J. Karhumäki and N. Ollinger, The commutation of finite sets: a challenging problem, Theoret. Comput. Sci. 273 (1-2) (2002), 69–79.

[Co71] J. H. Conway, *Regular Algebra and Finite Machines*, Chapman Hall (1971).

[CS78] K. Culik and A. Salomaa, On the decidability of homomorphism equivalence for languages, J. Comput. Systems Sci. 17 (1978), 241–250.

[Gr68] T. V. Griffiths, The unsolvability of the equivalence problem for λ-free nondeterministic generalized machines, J. Assoc. Comput. Mach. 15, 1968, 409–413.

[HH99] V. Halava and T. Harju, Undecidability of the equivalence of finite substitutions on regular language, Theoret. Informat. and Appl. 33, 1999, 117–124.

[HU79] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979.

[Ib78] O. Ibarra, The unsolvability of the equivalence problem for ε-free NGSM's with unary input (output) alphabet and applications, SIAM J. Comput. 7, 1978, 524–532.

[K85] J. Karhumäki, Problem P 97, Bull. EATCS 25, 1985, 185.

[KK85] J. Karhumäki and H. C. M. Kleijn, On the equivalence of composition of morphisms and inverse morphisms on regular languages, RAIRO Theoret. Informatics 19 (1985), 203–211.

[KL83] J. Karhumäki and M. Linna, A note on morphic characterization of languages, Discret. Appl. Math. 5 (1983), 243–246.

[KL99] J. Karhumäki and L. Lisovik, On the equivalence of finite substitutions and transducers, in: J. Karhumäki, H. Maurer, G. Paun and G. Rozenberg (eds), Jewels are Forever, Springer, Berlin, 1999, 97–108.

[KL03a] J. Karhumäki and L. P. Lisovik, A simple undecidable problem: The inclusion problem for finite substitutions on ab^*c, Inf. and Comput. 187, 2003, 40–48.

[KL03b] J. Karhumäki and L. P. Lisovik, The equivalence problem of finite substitutions on ab^*c, with applications, Intern. J. Found. Comput. Sci. 14, 2003, 699–710.

[KLP03] J. Karhumäki, A. Latteux and I. Petre, The commutation with codes and ternary sets of words, Proceedings of STACS'03, Lecture Notes in Comput. Sci. 2607, 2003, 74–84.

[KLP05] J. Karhumäki, M. Latteux and I. Petre, Commutation with codes, Theoret. Comput. Sci. (to appear).

[KP02] J. Karhumäki and I. Petre, Conway's Problem for three-word sets, Theoret. Comput. Sci. 289/1 (2002), 705–725.

[KP04] J. Karhumäki and I. Petre, Two Problems on Commutation of Languages, in: G. Paun, G. Rozenberg and A. Salomaa (eds), Current Trends in Theoretical Computer Science, The Challences of the New Century, World Scientific, Singapore (2004), 477–494.

[Ku05] M. Kunc, The power of commuting with finite sets of words, Proceedings of STACS'05, Lecture Notes in Comput. Sci. (to appear).

[La86] J. Lawrence, The nonexistence of finite test set for set-equivalence of finite substitutions, Bull. EATCS 28, 1986, 34–37.

[Li79] L. P. Lisovik, The identity problem of regular events over cartesian product of free and cyclic semigroups, Doklady of Academy of Sciences of Ukraine 6, 1979, 410–413.

[Li91] L. P. Lisovik, An undecidability problem for countable Markov chains, Kibernetika 2, 1991, 1–8.

[Li97] L. P. Lisovik, The equivalence problem for finite substitutions on regular languages, Doklady of Academy of Sciences of Russia 357, 1997, 299–301.

[LL83] M. Latteux and J. Leguy, On the composition of morphisms and inverse morphisms, Lecture Notes in Comput. Sci. 154, Springer-Verlag, 1983, 420–432.

[Lo83] M. Lothaire, *Combinatorics on Words*, Addison-Wesley, Reading, MA., (1983).

[LT87] M. Latteux and P. Turakainen, A new normal form for compositions of morphisms and inverse morphisms, Math. Systems Theory 20, 1987, 261–271.

[Ma77] G. S. Makanin, The problem of solvability of equations in a free semigroups, Mat. Sb. 103, 1977, 147–236; Math. USSR Sb. 32, 1977, 129–198.

[Mi67] M. Minsky, Computation: Finite and Infinite Machines, Prentice Hall, Englewood Cliffs, N. J., 1967.

[Pe02] I. Petre, Commutation Problems on Sets of Words and Formal Power Series, PhD Thesis, University of Turku (2002).

[Pl04] W. Plandowski, Satisfiability of word equations with constants is in PSPACE, Journal of the ACM 51 (3), 2004, 483–496.

[Ra89] B. Ratoandromanana, Codes et motifs, RAIRO Inform. Theor. 23(4) (1989), 425–444.

[Sa73] A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.

[Tu88] P. Turakainen, On some transducer equivalence problems for families of languages, Intern. J. Comput. Math. 23, 1988, 99–124.

Universality and Cellular Automata

K. Sutner

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
sutner@cs.cmu.edu

Abstract. The classification of discrete dynamical systems that are computationally complete has recently drawn attention in light of Wolfram's "Principle of Computational Equivalence". We discuss a classification for cellular automata that is based on computably enumerable degrees. In this setting the full structure of the semilattice of the c.e. degrees is inherited by the cellular automata.

1 Intermediate Degrees and Computational Equivalence

One of the celebrated results of recursion theory in the 20th century is the positive solution to Post's problem: there are computably enumerable sets whose Turing degree lies strictly between $\emptyset$, the degree of any recursive set, and $\emptyset'$, the degree of the Halting set or any other complete computably enumerable set. The result was obtained independently and almost simultaneously by R. M. Friedberg and A. A. Muchnik, see [8,14]. The method used in their construction of an intermediate degree is remarkable since it departs significantly from earlier attempts by Post and others to obtain such degrees by imposing structural conditions such as simplicity, hyper-simplicity or hyper-hyper-simplicity on the corresponding c.e. sets, see [17,11] for background information. The conditions are chosen so as to clearly rule out decidability and the hope was they also might enforce incompleteness. Of course, a non-trivial existence proof is required for this approach to succeed. Unfortunately, these attempts failed quite dramatically. For example, it was shown by Dekker that there is a hyper-simple set in every non-recursive computably enumerable degree and hence in particular in the complete degree.

By contrast, the so-called priority method used in the Friedberg-Muchnik construction builds two c.e. sets A and B whose only tangible property is that they are incomparable with respect to Turing reductions. The method generalizes easily to a construction of infinitely many incomparable sets. Alas, the sets constructed via priority arguments appear somewhat ad hoc and artificial. It is therefore tempting to search for "natural" examples of intermediate degrees, examples that would presumably arise as a side-effect of a less complicated construction. By natural we here mean that the generating device should admit a very simple description as opposed to, say, invariance under automorphisms of the semilattice of c.e. degrees. Of course, there are well-known results to the effect that every c.e. degree appears in a certain mathematical context. For example, all c.e. sets are Diophantine and can thus be defined by an integer polynomial. Similarly, every c.e. set is Turing equivalent to a finitely axiomatizable theory and word problems in finitely presented groups may have arbitrary c.e. degree. But the point here is to obtain a specific example of an intermediate degree using a reasonably simple mechanism to do so. For example, an elementary cellular automaton would certainly provide an indisputably natural example for an intermediate degree. A candidate for such an elementary cellular automaton might be rule 30, see the comments in section 5.

However, it is not clear at present whether there is much hope to find such examples. Indeed, in [28] Wolfram introduces his "Principle of Computational Equivalence" (PCE) which suggests, among other things, that the search is futile: Wolfram asserts that a certain class of computational processes obeys a 0/1 law: these processes are either decidable or already computationally universal. The evidence that Wolfram adduces for this principle is directly relevant to the search of natural examples: a large collection of simulations on various discrete dynamical systems such as Turing machine, register machines, tag systems, rewrite systems, combinators, and cellular automata. It is pointed out in chapter 3 of [28], that in all these classes of systems there are examples that exhibit exceedingly complicated behavior (and presumably even computational universality) even when the system in question is surprisingly small and has a very short description.

The reference contains a particularly striking example for a universal system that nonetheless has a very simple description: elementary cellular automaton rule number 110. The local map of this automaton is given by the ternary boolean function $\rho(x, y, z) = (\overline{x} \wedge y \wedge z) \oplus y \oplus z$ where $\oplus$ denotes exclusive or. It requires significant effort to show that, using fairly complicated and large segments of bits as the basic building blocks, it is possible to simulate cyclic tag systems on this cellular automaton, thus proving universality. There is a noteworthy difference between this and earlier examples of computationally universal cellular automata: the required configurations for this argument do not have finite support but are ultimately periodic in both spatial directions. In fact, it is not hard to see that rule 110 produces no interesting orbits on configurations with finite support.

While we are mostly interested in cellular automata as possible sources of natural intermediate degrees, we will first describe the problem in the slightly more general setting of an effective dynamical system. As a general purpose tool to measure the complexity of such a system we adapt M. Davis's notion of universality of Turing machines, see [5,6], to avoid coding conventions. We then discuss how systems of intermediate degree of complexity appear in various contexts. Needless to say, all of these results are universal rather than specific: they are similar to Matiyasevic's characterization of the Diophantine equations mentioned above in that they show that all c.e. degrees appear in some context but do not produce concrete or natural examples. Proofs for the results quoted in this paper as well as technical details can be found in the references.

2 Effective Dynamical Systems, Universality and Classification

We consider effective dynamical systems of the form $\mathcal{C} = \langle\, \mathcal{C}, \rightarrow \rangle$. Here $\mathcal{C}$ is the space of *configurations*, and $\rightarrow$ is the "next configuration" relation. We write $\xrightarrow{*}$ for the transitive reflexive closure of $\rightarrow$. The configurations are required to be finitary objects such as words and admit natural codings as integers. The relation $\rightarrow$ has to be primitive recursive on the (codes of the) configurations. In fact, in all relevant examples $\rightarrow$ is primitive recursive uniformly in a parameter that describes the particular instance of the system. E.g., for cellular automata the next configuration relation is primitive recursive uniformly in the local map of the automaton. Elementary cellular automata in particular can be parameterized by an 8-bit rule number. It is clear that the systems discussed in Wolfram's book all fit this general pattern.

Computational universality is traditionally defined in terms of simulations. First, one fixes a coding and decoding function $\alpha : \mathbb{N} \rightarrow \mathcal{C}$ and $\beta : \mathcal{C} \rightarrow \mathbb{N}$. Second, one adopts a notion of termination for the effective dynamical system so that an orbit may or may not lead to a "halting" configuration. It has to be easily decidable, say, primitive recursive, whether a configuration is halting. If a halting configuration appears in the orbit of $\alpha(n)$ let Y be the first such configuration and interpret $\beta(Y)$ as the output of the computation by $\mathcal{C}$ on input n. It is clear that $\mathcal{C}$ computes only partial recursive functions and it is natural to consider the system complete if it can compute all such functions.

While this approach is perfectly adequate for Turing machines or register machines it becomes a bit more problematic in the realm of cellular automata. There is no clear natural notion of termination here and even the coding functions are not so obvious. We therefore sidestep the issue of simulations entirely and instead adapt Davis's definition for universality of a Turing machine. In [6] Davis suggests that a more robust measure for the complexity of a Turing machine is the Turing degree of the collection of its instantaneous descriptions that lead to a halting configuration. For a Turing machine M let $\mathsf{ID}_M = \{\, I \mid I \xrightarrow{*} J,\ J \text{ halting} \,\}$. Then the machine is *Davis-universal* if ID_M is complete c.e.. Thus, there are no coding functions that might contribute to the apparent complexity of the machine. It is easy to see that any classically universal Turing machine is also Davis-universal; however, the opposite is false: a Turing machine that erases its tape before halting has trivial input/output behavior but may still be Davis-universal. Surprisingly, it was shown by Davis that any total recursive function can be computed by a *stable* Turing machine: all its instantaneous descriptions lead to a halting configuration. Thus, a stable Turing machine is trivial in the sense of Davis-universality: ID_M comprises all instantaneous descriptions and is trivially decidable.

In the context of an effective dynamical system $\mathcal{C}$ it is thus natural to consider the *complete orbit*

$$\mathrm{Orb}_{\mathcal{C}} = \{\, (X, Y) \mid X \xrightarrow{*} Y \,\}$$

which is c.e. given our constraints on the next configuration relation. We can then use the Turing degree of $\mathrm{Orb}_{\mathcal{C}}$ as a measure of the complexity of $\mathcal{C}$. Alternatively, we can interpret the degree measure as a decision problem, the *Reachability Problem* for $\mathcal{C}$: given two configurations X and Y we wish to determine whether Y lies in the orbit of X: is $X \xrightarrow{*} Y$?

Let us now focus on cellular automata. Since we insist on configurations being finitary we need to constrain the full Cantor space Σ^∞ of all biinfinite words over alphabet Σ. To obtain a reasonable class of configurations $\mathcal{C} \subseteq \Sigma^\infty$ one should insist that $\mathcal{C}$ is shift-invariant and dense. Furthermore, $\mathcal{C}$ must be closed under continuous shift-invariant maps (i.e. the global maps of cellular automata). Lastly, in order to obtain a reasonable image of the dynamics of the map on the whole space we insist on *reflection*: whenever a configuration $X \in \mathcal{C}$ has a predecessor under $\to$ in Σ^∞ then it also has a predecessor in $\mathcal{C}$. The classical choice for such a space of configurations is $\mathcal{C}_{\mathsf{f}}$, the collection of all configurations with finite support (to obtain reflection one has to either insist on quiescence of the local map or interpret the notion of finite support appropriately). Another possibility is to consider spatially periodic configurations of the form ${}^{\omega}u^{\omega}$. These configurations are somewhat special in that they correspond to finite cellular automata with periodic boundary conditions and all orbits here are necessarily finite. On the other hand, the largest such configuration space is the collection of all recursive configurations, see [20] for a proof that reflection holds in this case.

However, the Cook-Wolfram proof of the universality of elementary cellular automaton rule 110 suggests to consider a much more narrow class of configurations. To this end, define a configuration to be *almost periodic* if it has the form $X = {}^{\omega}uwv^{\omega}$ where u, v and w are all finite words. Then the class $\mathcal{C}_{\mathsf{ap}}$ of all almost periodic configurations has all the properties from above. Note that reflection does not hold for configurations of the more restricted form ${}^{\omega}uwu^{\omega}$. Furthermore, it is not clear how to transfer the universality argument for rule 110 into this setting. More precisely, the Cook-Wolfram argument uses the infinitely many copies of u in ${}^{\omega}uwv^{\omega}$ to produce timing signals whereas the copies of v encode the cyclic tag system. Both together operate on the center part w where the actual simulation of the tag system takes place.

Given a space $\mathcal{C}$ of suitable configurations we can define the *degree classification* for cellular automata as follows. For any c.e. degree $\mathbf{d}$ let

$$\mathbb{C}_{\mathbf{d}} = \{\, \rho \mid \deg(\mathrm{Orb}_\rho) = \mathbf{d} \,\}.$$

where ρ denotes the local map of the cellular automaton. We note that this classification does not distinguish between the first three levels of the Wolfram classification in its formalization by Culik and Yu: these three levels are subsumed by $\mathbb{C}_{\emptyset}$. On the other hand, class $\mathbb{C}_{\emptyset'}$ comprises all computationally universal cellular automata. The following hierarchy theorem is established in [21,23] over $\mathcal{C}_{\mathsf{f}}$.

Theorem 1. *The degree classes $\mathbb{C}_{\mathbf{d}}$ are non-empty for every c.e. degree $\mathbf{d}$.*

The construction uses a simulation of a Turing machine that recognizes an c.e. set W of degree $\mathbf{d}$. As in the case of Davis-universality we have to contend with unintended instantaneous descriptions, i.e., instantaneous descriptions that do not occur in any actual computation of the Turing machine. Since it is undecidable whether, say, a state of the Turing machine appears in a computation this requires a somewhat more careful construction than usual. The notion of stability can be relaxed in this context to mean that unintended configurations only produce decidable orbits and thus do not alter the degree of Orb_ρ. Incidentally, in his original paper Davis uses a syntactic normal form for computable functions rather than a direct modification of Turing machines, see also [15] and [1] for similar arguments. A suitably modified Turing machine can then be simulated by a one-dimensional cellular automaton to establish the theorem, see [23] and [18] for details. The latter reference in particular contains a detailed discussion of the coding issues.

As a consequence of this result there is little hope to obtain a taxonomy of cellular automata based on a few simple classes. For example, it follows from Sack's density theorem that for any two cellular automaton ρ_1 and ρ_2 such that $\mathrm{Orb}_{\rho_1} <_T \mathrm{Orb}_{\rho_2}$ there exists a third cellular automaton σ of strictly intermediate complexity: $\mathrm{Orb}_{\rho_1} <_T \mathrm{Orb}_\sigma <_T \mathrm{Orb}_{\rho_2}$. It is well-known that the Σ_1-theory of the semilattice of c.e. degrees is decidable. However, the reason for this decidability result lies in the fact that any countable partial order can be embedded into the semilattice so that the relative computational strength of cellular automata is indeed arbitrarily complicated. The full theory of the semilattice of c.e. degrees is known to be undecidable, see [9]; in fact it is extraordinarily complicated: its degree is $\emptyset^{(\omega)}$.

While Reachability deals with forward orbits, another classical decision problem for cellular automata is the existence of predecessors: given Y, is there a configuration X such that $X \to Y$? Configurations that do not admit a predecessor are often referred to as a Garden-of-Eden. It is easy to see that for finite or almost periodic configurations the existence of a predecessor is easily solvable in polynomial time. However, in the two-dimensional case the existence of an arbitrary predecessor configuration, given a finite target configuration, is co-c.e.-complete. The same problem is c.e.-complete for finite configurations and one can show that a suitable choice of cellular automaton will produce a predecessor problem of arbitrary c.e. degree, see [21].

More complicated predecessor problems for dimension one appear when we enlarge the class of admissible configurations to all recursive configurations. In this case it is co-c.e.-complete to determine whether a given configuration has a predecessor for any non-surjective cellular automaton, see [23]. Of course, surjectivity itself is easily testable in polynomial time for one-dimensional cellular automata.

3 Testing Complexity

The heuristic classification of one-dimensional cellular automata due to Wolfram [26] is based on the visual inspection of a segment of the orbits of the automaton. For sufficiently simple cellular automata the classification is quite compelling, see the many examples in Wolfram's book. However, any attempt to formalize this process in general seems to lead to strong undecidability. For example, Culik and Yu translated the Wolfram classes into the following categories: all configurations evolve to a fixed point, all configurations evolve to a periodic configuration and all configurations have decidable orbits, plus the class of all remaining cellular automata. These classes are shown to be undecidable in [4] where the underlying space of configurations here is $\mathcal{C}_{\mathsf{f}}$. We mention reference [1] in passing for another objection to the Wolfram classification. Closer inspection of the low classes shows the following.

Theorem 2. *It is Π_2-complete to check whether a finite configuration evolves to a fixed point. The same holds true for evolution to a periodic configuration.*

Spatially periodic configurations of course always evolve to periodic configurations but even in this case it is co-c.e.-complete to test whether a fixed point is ultimately reached. Likewise it is Σ_2-complete to test whether the inevitable limit cycle has length $O(n^k)$ for some fixed k. Here n denotes the length of the periodic configuration, see [19].

Unsurprisingly, it is even more difficult to determine the type of a cellular automaton in the degree classification.

Theorem 3. *For any c.e. degree* $\mathbf{d}$ *it is* $\Sigma_3^{\mathbf{d}}$*-complete to determine whether a cellular automaton belongs to class* $\mathbb{C}_{\mathbf{d}}$.

Similar results hold, *mutatis mutandis*, for the analogous cumulative hierarchies $\mathbb{C}_{\leq \mathbf{d}}$ and $\mathbb{C}_{\geq \mathbf{d}}$, see [23]. For example, $\mathbb{C}_{\leq \mathbf{d}}$ is $\Sigma_3^{\mathbf{d}}$-complete for all $\mathbf{d} < \emptyset'$, but $\mathbb{C}_{\leq \emptyset'}$ comprises all cellular automata and is thus trivial. It follows that it is Σ_3-complete to determine whether all orbits of a cellular automaton are decidable. However, testing whether the orbits are c.e.-complete is a Σ_4-complete task. In light of these undecidability results it would be desirable to develop a collection of sufficient conditions for universality. For example, for certain variants of the Game-of-Life there appears to be hope to identify the key mechanisms that make some of these automata universal, see [7]. Needless to say, it will be much harder to find sufficient criteria for properties that prevent universality without trivializing the orbits, but see the comments in section 5.

Another source of undecidability is the choice of appropriate backgrounds ${}^{\omega}uv^{\omega}$ in an almost periodic configuration ${}^{\omega}uwv^{\omega}$. As an example, consider again the universal elementary cellular automaton rule 110. If both u and v have length 1 then the orbit of any almost periodic configuration ${}^{\omega}uwv^{\omega}$ is trivially decidable. It is not hard to check that the same is true for slightly larger background patterns u and v. On the other hand, for sufficiently long background patterns the orbits become undecidable. There is no algorithmic way to determine the threshold between the two types of behavior.

Theorem 4. *Given a cellular automaton it is undecidable whether orbits on almost periodic configurations are undecidable for sufficiently long background patterns.*

For rule 110 it is the case that c.e.-complete orbits appear for background of sufficient length. However, in general this property is undecidable, see [22]. Furthermore, one can construct cellular automata whose Reachability Problem is undecidable on $\mathcal{C}_{\mathsf{ap}}$ but whose orbits on backgrounds of any fixed size is always decidable.

4 The Reversible Case

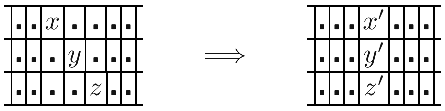
As we have seen, arbitrary cellular automata can have orbits of every c.e. degree. It is thus natural to search for restricted classes of cellular automata which may have less complicated orbits. One natural choice is the class of reversible cellular automata. The degree classification of the reversible cellular automata is indeed somewhat less complicated than for arbitrary cellular automata in the following sense. Consider the *Confluence Problem* for a dynamical system: given two configurations X and Y, is there a configuration Z that is reachable from both X and Y? In other words, do X and Y lead to the same limit cycle? It is clear that the Confluence Problem, just like Reachability is always c.e. The following result was established in [23].

Theorem 5. *Given any two c.e. degrees* $\mathbf{d}_1$ *and* $\mathbf{d}_2$ *there is an cellular automaton whose Reachability Problem has degree* $\mathbf{d}_1$ *and whose Confluence Problem has degree* $\mathbf{d}_2$*.*

It is clear that no analogous result can hold for reversible systems: X and Y here are confluent only in the trivial case where one configuration is reachable from the other.

The classical reference for reversible computation in the context of the mathematical theory of computation, rather than considerations more closely related to the physics of computation, is Bennett's paper that shows that arbitrary partial recursive functions can be computed reversibly on a suitable Turing machine, see [2]. In the construction, the intended output is copied before the computation is undone using an appropriate history tape. As a consequence, one can compute $\langle x, f(x) \rangle$ reversibly given any partial recursive function f. Somewhat surprisingly, the cost in terms of increased time and space complexity of the computation can be made to be quite modest in Bennett's construction, see [3]. It is also noteworthy that a decade prior to Bennett's paper Lecerf used reversible computation without generating output to establish the undecidability of certain equations, see [10]. Lecerf's construction carries over more naturally to the setting of cellular automata where input/output behavior is problematic. At any rate, for any c.e. set W there is a reversible Turing machine that accepts W.

Reversibility in the context of cellular automata is well-studied, see for example [25] for an overview. In [13,12] Morita and Harao gave an elegant construction for a reversible one-dimensional cellular automaton that is computationally universal, showing that the Lecerf-Bennett approach can be carried over into the realm of cellular automata. The construction allows one to build one-dimensional reversible cellular automata with relatively little effort. Their argument uses a three-track automaton whose global map is given by the composition of a shearing transformation (the top track moves to the left by one cell while the bottom track moves to the right) followed by the pointwise application of a map $f : \Sigma \to \Sigma$. More precisely, locally a configuration changes as follows. First the shearing transformation is applied to align x, y and z in one cell. Then (x, y, z) is replaced by $(x', y', z') = f(x, y, z)$.



The global map of the cellular automaton is then reversible if, and only if, the local map f is so reversible. In fact, one can even avoid a complete definition of f so long as the defined part does not include any non-injective assignments. At any rate, one has the following result for $\mathcal{C}_{\mathsf{f}}$, see [24].

Theorem 6. *For every c.e. degree* $\mathbf{d}$ *the degree class* $\mathbb{C}_{\mathbf{d}}$ *contains a reversible cellular automaton.*

The construction combines the standard stability trick with reversibility and a suitable simulation by a reversible cellular automaton ρ. The crux of the simulation is again to ensure that unintended configurations do not alter the degree of ρ. Since the cellular automaton is required to be reversible, no simple erasure technique is applicable. Instead, we exploit the upper and lower tracks to carry additional signal bits so that a cell contains three symbols (x_u, y, z_v) where $u, v \in \{0, 1\}$. As long as the local replacements correspond to appropriate actions of the underlying Turing machine the signal bits remain unchanged. If an undesirable event such as the collision of the two tape-heads occurs, the signal bits between the top and bottom-track are interchanged. Initially, in a finite starting configuration all bits in the quiescent part of the top track are 0 and 1 in the bottom track. For orbits that do not correspond to a computation of the Turing machine a signal bit will ultimately escape into the quiescent part and thereby provide a time-stamp for the configuration, which time-stamp renders the whole orbit decidable.

It is not clear whether this approach can be carried over to configuration spaces that lack a "quiescent" part such as recursive configurations: unintended local interactions here could appear in unboundedly many places and there seems to be no obvious way to construct time-stamps as in the finite or almost periodic case.

5 Conclusion

We have seen that intermediate c.e. degrees appear in many places in the study of the computational complexity of cellular automata, albeit in the form of uniform results: all c.e. degrees appear as the complexity of some decision problem or other associated with the automata. As regards the existence of a natural example of a specific intermediate degree the situation is much more difficult though perhaps not entirely hopeless. In a slightly different context H. Friedman has suggested on FOM, a mailing list for the foundations of mathematics, see http://www.cs.nyu.edu/mailman/listinfo/fom, that natural intermediate degrees might appear in the form of the theory of a single first-order formula φ in the language $\mathcal{L}(R)$ where R is a single binary relation symbol. Specifically Friedman is interested in the number of quantifiers needed in φ to ensure that the degree of $\mathsf{Th}(\varphi) = \{\, \psi \in \mathcal{L}(R) \mid \varphi \vdash \psi \,\}$ is intermediate. The conjecture is that 8 quantifiers might suffice. By the same token, considering sufficiently simple formulae of Peano arithmetic might produce a version of Wolfram's PCE; to wit, $\mathsf{Th}(\varphi)$ would appear to have degree $\emptyset$ or $\emptyset'$ for all formulae of size no more than 20.

It is unclear how cellular automata and their orbits relate to the notion of a "process" in Wolfram's PCE. One plausible objection against the use of intermediate degrees as a counterargument to PCE is that the construction relies heavily on information hiding. Indeed, in the standard Friedberg-Muchnik construction of two incomparable c.e. degrees the two set A and B so obtained have the property of being low, but their disjoint union is complete, see [16]. Thus, if one were to view the construction as a whole as a process then indeed this process would be computationally universal. Orbits of cellular automata would seem to provide little opportunity for information hiding, but the general Reachability problem may not be the right tool to access the information.

If one is willing to adopt different notions of reducibility other lines of inquiry become available. Recent work by Simpson has shown that if one adopts Muchnik degrees as the framework there are natural intermediate degrees. Interestingly, one of these natural examples is based on random reals. One should note that at least one elementary cellular automaton, known as rule 30, exhibits strong pseudo-random behavior and is in fact used as the default random number generator in the computer algebra system Mathematica, see [27]. It is tempting to speculate that the classification of the orbits of rule 30, on sufficiently general types of configurations, might provide another natural source of intermediate behavior. In particular almost periodic configurations might suffice for this purpose.

References

1. J. T. Baldwin and S. Shelah. On the classifiability of cellular automata. *Theoretical Computer Science*, 230(1-2):117–129, 2000.
2. C. H. Bennett. Logical reversibility of computation. *IBM journal of Research and Development*, pages 525–532, 1973.
3. C. H. Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, pages 766–776, 1989.
4. K. Culik and Sheng Yu. Undecidability of CA classification schemes. *Complex Systems*, 2(2):177–190, 1988.

5. M. Davis. *A note on universal Turing machines*, pages 167–175. Princeton University Press, 1956.
6. M. Davis. The definition of universal Turing machines. *Proc. of the American Mathematical Society*, 8:1125–1126, 1957.
7. K. M. Evans. Is Bosco's rule universal? In *MCU'04*, Sankt Petersburg, 2004.
8. R. M. Friedberg. Two recursively enumerable sets of incomparable degrees of unsolvability. *Proc. Natl. Acad. Sci. USA*, 43:236–238, 1957.
9. L. Harrington and S. Shelah. The undecidability of the recursively enumerable degrees. *Bull. Amer. Math. Soc.*, 6:79–80, 1982.
10. Y. Lecerf. Machine de Turing réversible. Insolubilité récursive en $n \in N$ de l'équation $u = \theta^n u$, où θ est un "isomorphisme de codes". *C. R. Acad. Sci. Paris*, 257:2597–2600, 1963.
11. M. Lerman. *Degrees of Unsolvability.* Perspectives in Mathematical Logic. Springer Verlag, 1983.
12. K. Morita. Reversible cellular automata. *J. Information Processing Society of Japan*, 35:315–321, 1994.
13. K. Morita and M. Harao. Computation universality of 1 dimensional reversible (injective) cellular automata. *Transactions Institute of Electronics, Information and Communication Engineers, E*, 72:758–762, 1989.
14. A. A. Muchnik. On the unsolvability of the problem of reducibility in the theory of algorithms. *Dokl. Acad. Nauk SSSR*, 108:194–197, 1956.
15. J. C. Shepherdson. Machine configuration and word problems of given degree of unsolvability. *Z. f. Math. Logik u. Grundlagen d. Mathematik*, 11:149–175, 1965.
16. R. I. Soare. The Friedberg-Muchnik theorem re-examined. *Canad. J. Math.*, 24:1070–1078, 1972.
17. R. I. Soare. *Recursively Enumerable Sets and Degrees.* Perspectives in Mathematical Logic. Springer Verlag, 1987.
18. K. Sutner. A note on Culik-Yu classes. *Complex Systems*, 3(1):107–115, 1989.
19. K. Sutner. Classifying circular cellular automata. *Physica D*, 45(1–3):386–395, 1990.
20. K. Sutner. De Bruijn graphs and linear cellular automata. *Complex Systems*, 5(1):19–30, 1991.
21. K. Sutner. Cellular automata and intermediate reachability problems. *Fundamentae Informaticae*, 52(1-3):249–256, 2002.
22. K. Sutner. Almost periodic configurations on linear cellular automata. To appear, 2003.
23. K. Sutner. Cellular automata and intermediate degrees. *Theoretical Computer Science*, 296:365–375, 2003.
24. K. Sutner. The complexity of reversible cellular automata. *Theoretical Computer Science*, 325(2):317–328, 2004.
25. T. Toffoli and N. Margolus. Injective cellular automata. *Physica D*, 45(1–3):386–395, 1990.
26. S. Wolfram. Computation theory of cellular automata. *Comm. Math. Physics*, 96(1):15–57, 1984.
27. S. Wolfram. *The Mathematica Book.* Cambridge University Press, 2002.
28. S. Wolfram. *A New Kind of Science.* Wolfram Media, 2002.

Leaf Language Classes
A Survey

Klaus W. Wagner

Institut für Informatik
Julius-Maximilians-Universität Würzburg
wagner@informatik.uni-wuerzburg.de

Abstract. The theory of leaf language classes is a fruitful field of research which has been developed since the beginning of the nineties. The leaf language model, in which one language (or a pair of languages) defines a class of languages, allows a uniform definition and treatment of many complexity classes. The results of this area give new insights into the structure of complexity classes and their relation to other fields of Theoretical Computer Science.

1 Introduction

Complexity classes based on nondeterministic computations have been studied since the beginning of the theory of compuational complexity. Such a complexity class is defined by a class of nondeterministic machines (e.g., nondeterministic polynomial time Turing machines) and a notion of acceptance. Since a nondeterministic machine can have many computation paths on a given input, the notion of acceptance has to make precise how the outcomes of these computation paths determine whether the input is accepted or not. In the simplest case of nondeterministic acceptance there has to be at least one accepting computation path, in the case of probabilistic acceptance there have to be more accepting paths than rejecting paths. During the last forty years more and more different and sophisticated notions of acceptance have been introduced and investigated.

Only at the beginning of the nineties a convincing step to unify these many different approaches was made. Bovet, Crescenzi, and Silvestri [BCS91, BCS92] and independently Vereshchagin [Ve93] developed a method to define notions of acceptance for nondeterministic machines which became later on known as the leaf language approach. (In [Vo03] it is mentioned that Papadimitriou and Sipser used this method in their lectures on complexity theory already around 1979.)

The essence of this method can be desribed easily. Consider a nondeterministic polynomial time Turing machine M. On input x, every computation path of M produces a symbol from a finite alphabet Σ. Considering the computation paths of M on x in lexicographical order the produced symbols build a finite sequence $\beta_M(x) \in \Sigma^*$, the *leaf word of M on x*. A language $L \subseteq \Sigma^*$ defines the *leaf language class* $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L)$ of all languages A for which there exists a nondeterministic polynomial time Turing machine M such that $x \in A \Leftrightarrow \beta_M(x) \in L$

for every x. In this sense one obtains immediately $\mathrm{NP} = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(0^*1(0 \cup 1)^*)$ and $\mathrm{PP} = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(\{x : x \in \{0,1\}^*$ and there are more 1's than 0's in $x\})$. Also less obvious characterizations of complexity classes like $\mathrm{P}^{\mathrm{NP}} = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}((0 \cup 1 \cup 2)^*10^*)$ can be proven.

Restricting in the above definiton the class of machines to such with balanced computation trees one obtains the *balanced leaf language classes* $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L)$ which might differ from the unbalanced classes as can be exemplified by $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(\mathrm{L_{mid}}) = \mathrm{P}$ and $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(\mathrm{L_{mid}}) = \mathrm{P}^{\mathrm{PP}}$ where $\mathrm{L_{mid}} =_{\mathrm{def}} \{x1y : x, y \in \{0,1\}^* \wedge |x| = |y|\}$. Restricting the class of machines to such with leaf words from a given language one can describe so-called *promise complexity classes* like BPP and UP. In fine, it turns out that practically every complexity class considered so far can be described by leaf languages.

The leaf language approach to define complexity classes has proven to be very fruitful over the last dozen of years. Here are some facts:

– All leaf language classes share some common properties. To prove these properties for a given class it is sufficient to show that this class can be described by a leaf language.
– The inclusion between two leaf language classes under every relativization is equivalent to a certain type of reducibility between the corresponding leaf languages. This can be used to shorten dramatically many oracle separation proofs in complexity theory.
– The study of classes defined by regular leaf languages has shown some very unexpected and interesting connections between the algebraic properties of leaf languages and the position of the corresponding leaf language classes in the landscape of complexity classes.

This paper surveys the (in my understanding) most important results on leaf language classes. However, a certain restriction is enforced here by the page restriction of this paper. It should be mentioned that [Vo99] and [Vo03] are further survey papers on leaf language classes. They differ from our survey in the choice and presentation of the results. Also, Heribert Vollmer's Leaf Language Homepage http://www.thi.uni-hannover.de/forschung/leafl/ is a valuable source for this field of research.

In Section 2 we introduce the complexity theoretic notations used in this paper. In Section 3 we give the main definitions on leaf language classes, we give some examples of leaf language characterizations of well-known complexity classes, and we state some basic properties of leaf language classes. In Section 4 we consider the family of all leaf language classes and its structure. Characterizations of this family and some of its subfamilies are given. For example, the family of all classes $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L)$ coincides with the family of all classes which are closed under polynomial time many-one reducibility and join and which have $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete languages. Further, we give an impression how big the variety of leaf language classes is: Every class of the boolean hierarchy, the polynomial hierarchy and the modulo hierarchy over a leaf language class is again a leaf language class.

In Section 5 we present results on classes with regular leaf languages. It turns out that there are interesting connections between the algebraic properties of leaf languages and the position of the corresponding leaf language classes (complexity classes) within PSPACE. In particular, the classes of the dot-depth hierarchy on the leaf language side correspond to the classes of the polynomial time hierarchy on the side of the leaf language classes. The smallest classes definable by regular leaf languages are exhibited. In Section 6 other classes of leaf languages are considered, in particular time, space, and circuit complexity classes.

In Section 7 Bovet, Crescenzi, Silvestri, and Vereshchagin's pioneering result on relativized leaf language classes is given. The inclusion between two balanced leaf language classes under every relativization is equivalent to the so-called polylogtime many-one reducibility between the corresponding leaf languages. A similar result for unbalanced leaf language classes is presented here for the first time.

In Section 8 we consider results on a leaf language model based on logarithmic space rather than polynomial time and on a leaf language model which defines classes of functions rather than classes of languages. In Section 9 we present a new extension of the leaf language model by which recursion theoretic classes can be defined by simple leaf languages, for example the classes of the arithmetic hierarchy. In Section 10 we discuss other approaches to define complexity classes, and we compare them with the leaf language approach.

2 Notations of Complexity Classes

Since many different complexity classes are used in this paper we will give here a brief summary of the corresponding notations.

For any function $t : \mathbb{N} \to \mathbb{N}$ such that $t(n) \geq n$, let DTIME(t) and NTIME(t) be the classes of languages which can be accepted by deterministic and nondeterministic, resp., Turing machines within time t. For any function $s : \mathbb{N} \to \mathbb{N}$ such that $s(n) \geq \log n$, let DSPACE(s) and NSPACE(s) be the classes of languages which can be accepted by deterministic and nondeterministic, resp., Turing machines within space s. For classes $\mathcal{F} \subseteq \mathbb{N}^{\mathbb{N}}$ define $\mathrm{DTIME}(\mathcal{F}) =_{\mathrm{def}} \bigcup_{t\in\mathcal{F}} \mathrm{DTIME}(t)$, $\mathrm{NTIME}(\mathcal{F}) =_{\mathrm{def}} \bigcup_{t\in\mathcal{F}} \mathrm{NTIME}(t)$, $\mathrm{DSPACE}(\mathcal{F}) =_{\mathrm{def}} \bigcup_{s\in\mathcal{F}} \mathrm{DSPACE}(s)$, and $\mathrm{NSPACE}(\mathcal{F}) =_{\mathrm{def}} \bigcup_{s\in\mathcal{F}} \mathrm{NSPACE}(s)$. Define $\mathrm{Pol} =_{\mathrm{def}} \{n^k : k \geq 1\}$ and $2^{\mathrm{Pol}} =_{\mathrm{def}} \{2^{n^k} : k \geq 1\}$. In particular, we set $\mathrm{P} =_{\mathrm{def}} \mathrm{DTIME(Pol)}$, $\mathrm{NP} =_{\mathrm{def}} \mathrm{NTIME(Pol)}$, $\mathrm{L} =_{\mathrm{def}} \mathrm{DSPACE(log)}$, $\mathrm{NL} =_{\mathrm{def}} \mathrm{NSPACE(log)}$, and $\mathrm{PSPACE} =_{\mathrm{def}} \mathrm{DSPACE(Pol)}$. For functions $f, g \in \mathbb{N}$ define $(f \circ g)(n) =_{\mathrm{def}} f(g(n))$ for $n \in \mathbb{N}$, and for classes $\mathcal{F}, \mathcal{G} \subseteq \mathbb{N}^{\mathbb{N}}$ define $\mathcal{F} \circ \mathcal{G} =_{\mathrm{def}} \{f \circ g : f \in \mathcal{F} \wedge g \in \mathcal{G}\}$.

The class PP is defined as the class of languages which can be accepted by probabilistic polynomial time Turing machines, and the class BPP is defined as the class of languages which can be accepted by probabilistic polynomial time Turing machines with bounded error probability. If we consider the number of accepting paths rather than the acceptance probability then this does not change the class PP but in the bounded case we obtain the class $\mathrm{BPP}_{\mathrm{path}} \supseteq \mathrm{BPP}$.

For a class $\mathcal{K}$ accepted in a certain way by machines of a certain type and a language X, the *relativized class* $\mathcal{K}^X$ is the class of languages accepted in the same way by machines of the same type which in addition have access to X as an oracle. Note that this general definition needs some more explanation in special cases. For a class $\mathcal{M}$ of languages set $\mathcal{K}^{\mathcal{M}} =_{\text{def}} \bigcup_{X \in \mathcal{M}} \mathcal{K}^X$. In particular, $P^{\mathcal{M}}$ is the class of languages which can be accepted by deterministic Turing machines in polynomial time with an oracle from $\mathcal{M}$. Let further $P^{\mathcal{M}}[1]$ be the class of languages which can be accepted by deterministic Turing machines in polynomial time with one query to an oracle from $\mathcal{M}$.

Let $A \triangle B =_{\text{def}} (A \setminus B) \cup (B \setminus A)$ denote the *symmetric difference* of the sets A and B. Let A, B be languages over the finite alphabet Σ, and let a and b be different symbols from Σ. The set $A \uplus B =_{\text{def}} aA \cup bB$ is said to be the *join* of the sets A and B.

Now we consider special operations on language classes. Let $\mathcal{K}$ and $\mathcal{M}$ be classes of languages, and let $k \geq 2$. We define

$$\begin{aligned}
\text{co-}\mathcal{K} &=_{\text{def}} \{\overline{A} : A \in \mathcal{K}\},\\
\mathcal{K} \wedge \mathcal{M} &=_{\text{def}} \{A \cap B : A \in \mathcal{K} \text{ and } B \in \mathcal{M}\},\\
\mathcal{K} \oplus \mathcal{M} &=_{\text{def}} \{A \triangle B : A \in \mathcal{K} \text{ and } B \in \mathcal{M}\},\\
\mathcal{K} \nabla \mathcal{M} &=_{\text{def}} \{A : \exists D(D \in \mathrm{P} \wedge A \cap D \in \mathcal{K} \wedge A \cap \overline{D} \in \mathcal{M})\},\\
\exists\cdot\mathcal{K} &=_{\text{def}} \{A : \text{there exists a polynomial } p \text{ and a } B \in \mathcal{K} \text{ such that}\\
&\qquad \forall x(x \in A \leftrightarrow \exists z(|z| = p(|x|) \wedge (x,z) \in B))\},\\
\forall\cdot\mathcal{K} &=_{\text{def}} \{A : \text{there exists a polynomial } p \text{ and a } B \in \mathcal{K} \text{ such that}\\
&\qquad \forall x(x \in A \leftrightarrow \forall z(|z| = p(|x|) \rightarrow (x,z) \in B))\},\\
\exists!\cdot\mathcal{K} &=_{\text{def}} \{A : \text{there exists a polynomial } p \text{ and a } B \in \mathcal{K} \text{ such that}\\
&\qquad \forall x(x \in A \leftrightarrow |\{z : |z| = p(|x|) \wedge (x,z) \in B)\}| = 1)\},\\
\mathrm{U}\cdot\mathcal{K} &=_{\text{def}} \{A : \text{there exists a polynomial } p \text{ and a } B \in \mathcal{K} \text{ such that}\\
&\qquad \forall x(x \in A \rightarrow |\{z : |z| = p(|x|) \wedge (x,z) \in B\}| = 1) \text{ and}\\
&\qquad \forall x(x \notin A \rightarrow |\{z : |z| = p(|x|) \wedge (x,z) \in B\}| = 0)\},\\
\mathrm{Mod}_k\cdot\mathcal{K} &=_{\text{def}} \{A : \text{there exists a polynomial } p \text{ and a } B \in \mathcal{K} \text{ such that}\\
&\qquad \forall x(x \in A \leftrightarrow |\{z : |z| = p(|x|) \wedge (x,z) \in B\}| \equiv 0(k))\}.
\end{aligned}$$

In particular, define 1-NP $=_{\text{def}} \exists!\cdot$P, UP $=_{\text{def}}$ U$\cdot$P, Mod$_k$P $=_{\text{def}}$ Mod$_k\cdot$P for $k \geq 2$, and $\oplus$P $=_{\text{def}}$ Mod$_2\cdot$P.

The *boolean hierarchy* over a language class $\mathcal{K}$ is the smallest family of language classes that contains $\mathcal{K}$ and that contains with the classes $\mathcal{M}$ and $\mathcal{L}$ also the classes co-$\mathcal{M}$ and $\mathcal{M} \wedge \mathcal{L}$. If $\mathcal{K}$ is closed under union and intersection then every class of the boolean hierachy over $\mathcal{K}$ coincides with one of the classes $\mathcal{K}(k)$ or co-$\mathcal{K}(k)$ where $\mathcal{K}(1) =_{\text{def}} \mathcal{K}$ and $\mathcal{K}(k+1) =_{\text{def}} \mathcal{K}(k) \oplus \mathcal{K}$ for $k \geq 1$. Notice that $\mathcal{K}(k+1) = \text{co-}\mathcal{K}(k) \wedge \mathcal{K}$ for $k \geq 1$. Let BH $=_{\text{def}} \bigcup_{k \geq 1} \mathrm{NP}(k)$.

The *polynomial hierarchy* over a language class $\mathcal{K} \supseteq$ P is the smallest family of language classes that contains $\mathcal{K}$ and that contains with the class $\mathcal{M}$ also the classes co-$\mathcal{M}$, $\exists\cdot\mathcal{M}$, and $\forall\cdot\mathcal{M}$. We define $\Sigma_0^{\mathrm{p}} =_{\text{def}} \Pi_0^{\mathrm{p}} =_{\text{def}} \Delta_0^{\mathrm{p}} =_{\text{def}} \mathrm{P}$, $\Delta_{k+1}^{\mathrm{p}} =_{\text{def}} \mathrm{P}^{\Sigma_k^{\mathrm{p}}}$, $\Sigma_{k+1}^{\mathrm{p}} =_{\text{def}} \exists\cdot\Pi_k^{\mathrm{p}}$, $\Pi_{k+1}^{\mathrm{p}} =_{\text{def}} \forall\cdot\Sigma_k^{\mathrm{p}}$ for $k \geq 0$, and PH $=_{\text{def}} \bigcup_{k \geq 0}(\Sigma_k^{\mathrm{p}} \cup \Pi_k^{\mathrm{p}} \cup \Delta_k^{\mathrm{p}})$. This hierarchy is called the *polynomial time hierarchy.*

The *modulo hierarchy* over a language class $\mathcal{K} \supseteq$ P is the smallest family of language classes that contains $\mathcal{K}$ and that contains with the class $\mathcal{M}$ also the classes co-$\mathcal{M}$, $\exists\cdot\mathcal{M}$, $\forall\cdot\mathcal{M}$, and $\mathrm{Mod}_k\cdot\mathcal{M}$ for every $k \geq 2$. Let MOD-PH be the union of all classes of the modulo hierarchy over P. Note that every class of the polynomial hierarchy over $\mathcal{K}$ is also a class of the modulo hierarchy over $\mathcal{K}$, and thus PH $\subseteq$ MOD-PH.

For $k \geq 1$, let Σ_k-LOGTIME be the class of all languages which can be accepted by a Σ_k-alternating Turing machine in logarithmic time. Such a machine accesses the input as an oracle, it starts with an existing configuration, and it alternates at most $k-1$ times on every computation path. Let $\mathrm{AC}^0 =_{\mathrm{def}} \bigcup_{k\geq 1} \Sigma_k$-LOGTIME. Alternatively, AC^0 is the class of all languages which can be accepted by uniform unbounded fan-in $\{\wedge, \vee, \neg\}$-circuits of polynomial size and constant depth. Let NC^1 be the class of all languages which can be accepted by an alternating Turing machine in logarithmic time. Alternatively, NC^1 is the class of all languages which can be accepted by uniform bounded fan-in $\{\wedge, \vee, \neg\}$-circuits of polynomial size and logarithmic depth.

The polynomial time many-one reducibility is denoted by $\leq_{\mathrm{m}}^{\mathrm{p}}$.

For more details on complexity classes the reader is referred to [BDG88], [BDG90], [Pa94], and [Aa04].

3 Leaf Language Classes, Examples and Basic Properties

In this section we introduce the notion of leaf language classes [BCS91, Ve93], we give some examples of leaf language characterizations of well-known complexity classes, and we state some basic properties of leaf language classes.

Let M be a nondeterministic polynomial time machine that, in every step, splits a computation path into at most two computation paths. Hence a computation path of M on input x can be described by a word from $\{0,1\}^*$. Let the nondeterministic polynomial time machine M produce on every computation path z on input x a symbol $M(x,z)$ from a finite alphabet Σ. Let $r_1, r_2, \ldots, r_k \in \{0,1\}^*$ be the computation paths of M on input x in lexicographical order ($\leq$). The *leaf word* of M on input x is defined as $\beta_M(x) =_{\mathrm{def}} M(x,r_1)M(x,r_2)\ldots M(x,r_k)$.

A machine is called *balanced* if for every input x there exists an $m \geq 0$ and an $r \in \{0,1\}^m$ such that $\{s : s \in \{0,1\}^m \wedge s \leq r\}\cdot\{0,1\} \cup \{s : s \in \{0,1\}^m \wedge s > r\}$ is the set of all computation paths of M on x.

We define the *leaf language classes* $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L_1|L_2)$ (unbalanced model) and $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_2)$ (balanced model) as follows. If L_1 and L_2 are languages over a finite alphabet Σ such that $L_1, L_2 \notin \{\emptyset, \{\varepsilon\}\}$ and $L_1 \cap L_2 = \emptyset$ then we call (L_1, L_2) a *pair of leaf languages*. For a pair (L_1, L_2) of leaf languages and a set $A \subseteq \Delta^*$ we define

$A \in \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L_1|L_2) \Leftrightarrow_{\mathrm{def}}$ there exists a nondeterministic polynomial time machine M such that, for every $x \in \Delta^*$, $x \in A \rightarrow \beta_M(x) \in L_1$ and $x \notin A \rightarrow \beta_M(x) \in L_2$,

$A \in \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_2) \Leftrightarrow_{\mathrm{def}}$ there exists a balanced nondeterministic polynomial time machine M such that, for every $x \in \Delta^*$, $x \in A \rightarrow \beta_M(x) \in L_1$ and $x \notin A \rightarrow \beta_M(x) \in L_2$.

If L is a language over a finite alphabet and Σ is the smallest alphabet such that $L \subseteq \Sigma^*$ then we define $\overline{L} =_{\mathrm{def}} \Sigma^* \setminus L$. If $L, \overline{L} \notin \{\emptyset, \{\varepsilon\}\}$ then we call L a *leaf language*, and we define $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L) =_{\mathrm{def}} \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L|\overline{L})$ and $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L) =_{\mathrm{def}} \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L|\overline{L})$. For a class $\mathcal{K}$ of languages we set $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(\mathcal{K}) =_{\mathrm{def}} \bigcup_{L\in\mathcal{K}} \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L)$ and $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(\mathcal{K}) =_{\mathrm{def}} \bigcup_{L\in\mathcal{K}} \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L)$. We use the following convention: If $\mathrm{Leaf}^{\mathrm{p}}(L_1|L_2)$ is used in a statement then this statement is valid for $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L_1|L_2)$ and $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_1|L_2)$, and if $\mathrm{Leaf}^{\mathrm{p}}(L)$ is used in a statement then this statement is valid for $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L)$ and $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L)$.

The classes of type $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_1|L_2)$ are also called *semantic classes* or *promise classes* (since every machine M accepting a language from $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_1|L_2)$ fulfills the promise that $\beta_M(x) \in L_1 \cup L_2$ for every input x), and the classes of type $\mathrm{Leaf}^{\mathrm{p}}(L)$ are also called *syntactic classes* or *complementary classes.*

As examples let us consider leaf language characterizations of well-known complexity classes. For $\varepsilon \in (0,1)$, let $\mathrm{L}_{<\varepsilon}$ ($\mathrm{L}_{>\varepsilon}$) be the set of all words from $\{0,1\}^*$ in which the number of 1's is not greater (not less, resp.) than $\varepsilon \cdot |x|$. Further, let $\mathrm{L}_{\mathrm{mid}} =_{\mathrm{def}} \{x1y : x, y \in \{0,1\}^* \wedge |x| = |y|\}$.

Theorem 1.
1. $\mathrm{Leaf}^{\mathrm{p}}(0^*1(0\cup 1)^*) = \mathrm{NP}$
2. $\mathrm{Leaf}^{\mathrm{p}}(0^*1(0\cup 1)^*2^*|\,0^*2^*3(2\cup 3)^*) = \mathrm{NP}\cap \text{co-NP}$
3. $\mathrm{Leaf}^{\mathrm{p}}(0^*10^*|\,0^*) = \mathrm{UP}$
4. $\mathrm{Leaf}^{\mathrm{p}}(0^*10^*|\,0^*20^*) = \mathrm{UP}\cap\text{co-UP}$
5. $\mathrm{Leaf}^{\mathrm{p}}(0^*10^*) = 1\text{-NP}$
6. $\mathrm{Leaf}^{\mathrm{p}}(00^*\cup 10^*1(0\cup 1)^*) = \mathrm{P}^{\mathrm{NP}}[1]$
7. $\mathrm{Leaf}^{\mathrm{p}}((0\cup 1\cup 2)^*10^*) = \mathrm{P}^{\mathrm{NP}}$
8. $\mathrm{Leaf}^{\mathrm{p}}((0^*10^*1)^*0^*) = \oplus\mathrm{P}$
9. $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}((11)^*) = \mathrm{P}$ *and* $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}((11)^*) = \oplus\mathrm{P}$
10. $\mathrm{Leaf}^{\mathrm{p}}(\mathrm{L}_{>1/2}) = \mathrm{PP}$
11. $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(\mathrm{L}_{\mathrm{mid}}) = \mathrm{P}$ *and* $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(\mathrm{L}_{\mathrm{mid}}) = \mathrm{P}^{\mathrm{PP}}$
12. $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(\mathrm{L}_{>2/3}|\mathrm{L}_{<1/3}) = \mathrm{BPP}$, $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(\mathrm{L}_{>2/3}|\mathrm{L}_{<1/3}) = \mathrm{BPP}_{\mathrm{path}}$.

The leaf words of a balanced polynomial time machine can also be given by two polynomial time computable functions. This provides the following characterization of $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_1|L_2)$.

Theorem 2. *Let* $(L_1, L_2) \subseteq (\Sigma^*)^2$ *be a pair of leaf languages, and let* $A \in \Delta^*$.
$A \in \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_1|L_2) \Leftrightarrow$ *there exist polynomial time computable* $h : \Delta^* \to \mathbb{N}$ *and* $g : \Delta^* \times \mathbb{N} \to \Sigma$ *such that for all* $x \in \Delta^*$,
$x \in A \to g(x,0)g(x,1)g(x,2)\ldots g(x,h(x)) \in L_1$ *and*
$x \notin A \to g(x,0)g(x,1)g(x,2)\ldots g(x,h(x)) \in L_2$.

The next theorem is about the relations between the balanced and the unbalanced leaf language classes. A pair $(L_1, L_2) \subseteq (\Sigma^*)^2$ of leaf languages is *paddable* if there exists an $a \in \Sigma$ such that for all $x, y \in \Sigma^*$ there holds $xy \in L_1 \leftrightarrow xay \in L_1$ and $xy \in L_2 \leftrightarrow xay \in L_2$. A leaf language $L \subseteq \Sigma^*$ is *paddable* if there exists an $a \in \Sigma$ such that for all $x, y \in \Sigma^*$ there holds $xy \in L \leftrightarrow xay \in L$.

Theorem 3. *Let (L_1, L_2) be a pair of leaf languages.*

1. $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_1|L_2) \subseteq \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L_1|L_2) \subseteq \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_1|L_2)^{\mathrm{PP}}$.
2. *If (L_1, L_2) is paddable then* $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_1|L_2) = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L_1|L_2)$.

The difference between the balanced and the unbalanced case can be really considerable. This can be the difference between P and P^{PP} as exemplified by Theorem 1.11. Notice that P^{PP} includes all the polyomial time hierarchy. This difference can be explained as follows: Given a bit position in $\beta_M(x)$, one can easily compute the computation path of a balanced machine M on x which produces this bit. For an unbalanced machine this is a counting problem, i.e., this needs oracle queries to a PP set. For other examples where the balanced and the unbalanced leaf language classes for the same leaf language differ see Theorem 1.9, Theorem 19 and the remark after Theorem 17.

The class P is the minimal class which can be described by leaf languages.

Proposition 1. *Let (L_1, L_2) be a pair of leaf languages.*

1. $\mathrm{Leaf}^{\mathrm{p}}(L_1|L_2) \supseteq \mathrm{P}$.
2. *If L_1 or L_2 is finite then* $\mathrm{Leaf}^{\mathrm{p}}(L_1|L_2) = \mathrm{P}$.

Problem. The regular languages L such that $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L) = \mathrm{P}$ and $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L) = \mathrm{P}$ are described in Corollary 2 and Corollary 3, resp. Theorem 1.11 shows that there exist non-regular languages L such that $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L) = \mathrm{P}$. Is there a non-regular language L such that $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L) = \mathrm{P}$? If not, are there non-regular languages L_1 and L_2 such that $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L_1|L_2) = \mathrm{P}$?

Proposition 2. [Bo04]

1. *If (L_1, L_2) is paddable then* $\mathrm{Leaf}^{\mathrm{p}}(L_1|L_2) \supseteq \mathrm{UP} \cap \text{co-UP}$.
2. *If L is paddable then* $\mathrm{Leaf}^{\mathrm{p}}(L) \supseteq \mathrm{UP}$ *or* $\mathrm{Leaf}^{\mathrm{p}}(L) \supseteq \text{co-UP}$.

Problem. Theorem 1.4 shows that Proposition 2.1 cannot be improved. On the other hand, we conjecture that Proposition 2.2 can be improved. This is because $\mathrm{Leaf}^{\mathrm{p}}(L) = \mathrm{UP}$ implies by Theorem 4.5 that UP has complete problems. However, there are oracles relative to which UP has no complete problems. Can UP and co-UP be replaced with larger classes in Proposition 2.2?

All leaf language classes share some basic properties (for the balanced case see [BCS91]). A class $\mathcal{K}$ of languages is *recursively presentable* if there exists a recursive set A such that $\mathcal{K} = \{\{x : (i, x) \in A\} : i \in \mathbb{N}\}$.

Theorem 4. *Let (L_1, L_2) be a pair of leaf languages, and let L be a leaf language.*

1. $\mathrm{Leaf}^{\mathrm{p}}(L_1|L_2)$ *is closed under $\leq^{\mathrm{p}}_{\mathrm{m}}$-reducibility.*
2. $\mathrm{Leaf}^{\mathrm{p}}(L_1|L_2)$ *is closed under join.*
3. $\mathrm{Leaf}^{\mathrm{p}}(L_1|L_2)$ *is countable.*
4. *If L_1 and L_2 are recursive then* $\mathrm{Leaf}^{\mathrm{p}}(L_1|L_2)$ *is recursively presentable.*
5. $\mathrm{Leaf}^{\mathrm{p}}(L)$ *has $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete sets.*

In fact, the leaf language classes of type $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L)$ are the only balanced leaf language classes which have $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete sets.

Theorem 5. [BS97] *Let (L_1, L_2) be a pair of (recursive) leaf languages. The class $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_2)$ has a $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete set if and only if there exists a (recursive) language L such that $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_2) = \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L)$.*

As a $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete language for $\mathrm{Leaf}^{\mathrm{p}}(L)$ a succinct version of L (via description by circuits) can be chosen. For the balanced case this is shown in [BL96] and [Ve98].

4 The Familiy of Leaf Language Classes

In this section we will get an impression about the structure of the most important families of leaf language classes. For a class $\mathcal{K}$ of languages we define $\mathcal{L}_{\mathrm{u}}^{\mathrm{p}}(\mathcal{K}) =_{\mathrm{def}} \{\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L_1|L_2) : (L_1, L_2) \in \mathcal{K}^2 \text{ is a pair of leaf languages}\}$ and the "complementary" family $\mathcal{C}_{\mathrm{u}}^{\mathrm{p}}(\mathcal{K}) =_{\mathrm{def}} \{\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L) : L \in \mathcal{K} \text{ is a leaf language}\}$. Let ALL, REC, and PAD be the classes of all, all recursive and all paddable, resp., languages. Set $\mathcal{L}_{\mathrm{u}}^{\mathrm{p}} =_{\mathrm{def}} \mathcal{L}_{\mathrm{u}}^{\mathrm{p}}(\mathrm{ALL})$ and $\mathcal{C}_{\mathrm{u}}^{\mathrm{p}} =_{\mathrm{def}} \mathcal{C}_{\mathrm{u}}^{\mathrm{p}}(\mathrm{ALL})$. In the same way we define $\mathcal{L}_{\mathrm{b}}^{\mathrm{p}}(\mathcal{K})$, $\mathcal{C}_{\mathrm{b}}^{\mathrm{p}}(\mathcal{K})$, $\mathcal{L}_{\mathrm{b}}^{\mathrm{p}}$, and $\mathcal{C}_{\mathrm{b}}^{\mathrm{p}}$. Because of Theorem 3 we have $\mathcal{L}_{\mathrm{u}}^{\mathrm{p}}(\mathrm{PAD}) = \mathcal{L}_{\mathrm{b}}^{\mathrm{p}}(\mathrm{PAD})$ and $\mathcal{C}_{\mathrm{u}}^{\mathrm{p}}(\mathrm{PAD}) = \mathcal{C}_{\mathrm{b}}^{\mathrm{p}}(\mathrm{PAD})$.

Converting in a sense Theorem 4 one gets complete characterizations of these families of balanced leaf language classes.

Theorem 6. [BS97]
1. $\mathcal{L}_{\mathrm{b}}^{\mathrm{p}} = \{\mathcal{K} : \mathcal{K}$ *is countable and is closed under join and* $\leq_{\mathrm{m}}^{\mathrm{p}}\}$
2. $\mathcal{C}_{\mathrm{b}}^{\mathrm{p}} = \{\mathcal{K} : \mathcal{K}$ *has* $\leq_{\mathrm{m}}^{\mathrm{p}}$*-complete sets and is closed under* $\leq_{\mathrm{m}}^{\mathrm{p}}\}$
3. $\mathcal{L}_{\mathrm{b}}^{\mathrm{p}}(\mathrm{REC}) = \{\mathcal{K} : \mathcal{K}$ *is recurs. presentable and is closed under join and* $\leq_{\mathrm{m}}^{\mathrm{p}}\}$
4. $\mathcal{C}_{\mathrm{b}}^{\mathrm{p}}(\mathrm{REC}) = \{\mathcal{K} : \mathcal{K}$ *has recurs.* $\leq_{\mathrm{m}}^{\mathrm{p}}$*-complete sets and is closed under* $\leq_{\mathrm{m}}^{\mathrm{p}}\}$

The following theorem clarifies the algebraic nature of some families of leaf language classes. Notice that, if $\mathcal{K}, \mathcal{M} \supseteq \mathrm{P}$ are closed under $\leq_{\mathrm{m}}^{\mathrm{p}}$, then $\mathcal{K} \nabla \mathcal{M} = \{A : \text{there exists } B \in \mathcal{K} \text{ and } C \in \mathcal{M} \text{ such that } A \leq_{\mathrm{m}}^{\mathrm{p}} B \uplus C\}$.

Theorem 7. *1.* [Bo94b] $(\mathcal{L}_{\mathrm{b}}^{\mathrm{p}}, \subseteq)$ *and* $(\mathcal{L}_{\mathrm{b}}^{\mathrm{p}}(\mathrm{REC}), \subseteq)$ *are distributive lattices.*
2. [BS97] $(\mathcal{C}_{\mathrm{b}}^{\mathrm{p}}, \subseteq)$ *and* $(\mathcal{C}_{\mathrm{b}}^{\mathrm{p}}(\mathrm{REC}), \subseteq)$ *are distributive upper semi-lattices.*
The least upper bound and the greatest lower bound (if any) are given by the operations ∇ and $\cap$, resp.

Leaf language classes of type $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_2)$ can be characterized as intersections of leaf language classes of type $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L)$.

Theorem 8. [BS97] $\mathcal{L}_{\mathrm{b}}^{\mathrm{p}} = \mathcal{C}_{\mathrm{b}}^{\mathrm{p}} \wedge \mathcal{C}_{\mathrm{b}}^{\mathrm{p}}$ *and* $\mathcal{L}_{\mathrm{b}}^{\mathrm{p}}(\mathrm{REC}) = \mathcal{C}_{\mathrm{b}}^{\mathrm{p}}(\mathrm{REC}) \wedge \mathcal{C}_{\mathrm{b}}^{\mathrm{p}}(\mathrm{REC})$

Problem. Are Theorem 6, Theorem 7, and Theorem 8 also valid for unbalanced leaf language classes?

The following theorem shows that many operations on language classes can be "modelled" with the corresponding leaf languages.

Theorem 9. *Let $(L_1, L_1') \subseteq (\Sigma_1^*)^2$ and $(L_2, L_2') \subseteq (\Sigma_2^*)^2$ be pairs of leaf languages such that $\Sigma_1 \cap \Sigma_2 = \emptyset$.*

1. $\text{co-Leaf}^{\text{p}}(L_1|L_1') = \text{Leaf}^{\text{p}}(L_1'|L_1)$.
2. [BCS91] $\text{Leaf}^{\text{p}}(L_1|L_1') \cap \text{Leaf}^{\text{p}}(L_2|L_2') = \text{Leaf}^{\text{p}}(L_1L_2|L_1'L_2')$.
3. $\text{Leaf}^{\text{p}}(L_1|L_1') \nabla \text{Leaf}^{\text{p}}(L_2|L_2') = \text{Leaf}^{\text{p}}(L_1 \cup L_2|L_1' \cup L_2')$ *and* $\text{Leaf}^{\text{p}}(L_1) \nabla \text{Leaf}^{\text{p}}(L_2) = \text{Leaf}^{\text{p}}(L_1 \uplus L_2)$.
4. $\text{Leaf}^{\text{p}}(L_1|L_1') \oplus \text{Leaf}^{\text{p}}(L_2|L_2') = \text{Leaf}^{\text{p}}(L_1L_2' \cup L_1'L_2|L_1L_2 \cup L_1'L_2')$.
5. $\text{P}^{\text{Leaf}^{\text{p}}(L_1|L_1')}[1] = \text{Leaf}^{\text{p}}(L_1|L_1') \oplus \text{P} = \text{Leaf}^{\text{p}}(L_1 \uplus L_1'|L_1' \uplus L_1)$ *and* $\text{P}^{\text{Leaf}^{\text{p}}(L_1)}[1] = \text{Leaf}^{\text{p}}(L_1) \oplus \text{P} = \text{Leaf}^{\text{p}}(L_1 \uplus \overline{L_1})$.

One can use the results of Theorem 9 and Theorem 6 to obtain the first two statements of the following theorem about closure properties of some families of leaf language classes. We define that the operation $\text{P}^{\bullet}$ applied to a class $\mathcal{K}$ results in $\text{P}^{\mathcal{K}}$, analogously for $\text{P}^{\bullet}[1]$.

Theorem 10. *1. If the class $\mathcal{K}$ of languages is closed under union, concatenation, and iteration then the classes $\mathcal{L}_{\text{b}}^{\text{p}}(\mathcal{K})$ and $\mathcal{L}_{\text{u}}^{\text{p}}(\mathcal{K})$ are closed under the operations* co-, $\oplus$, $\cap$, ∇, $\text{P}^{\bullet}[1]$, $\exists$, $\forall$, *and* Mod_k *for $k \geq 2$, and the classes $\mathcal{C}_{\text{b}}^{\text{p}}(\mathcal{K})$ and $\mathcal{C}_{\text{u}}^{\text{p}}(\mathcal{K})$ are closed under the operations* co-, ∇, $\text{P}^{\bullet}[1]$, $\exists$, $\forall$, *and* Mod_k *for $k \geq 2$. This applies, besides others, to $\mathcal{K} = \text{ALL}, \text{REC}, \text{REG}, \text{PAD}$.*
2. *The families $\mathcal{C}_{\text{b}}^{\text{p}}$ and $\mathcal{L}_{\text{b}}^{\text{p}}(\text{PAD})$ are closed under $\oplus$.*
3. [BCS91] *The families $\mathcal{L}_{\text{b}}^{\text{p}}$ and $\mathcal{C}_{\text{b}}^{\text{p}}$ are closed under $\text{P}^{\bullet}$.*
4. [Tr02] *If $\text{Leaf}_{\text{u}}^{\text{p}}(L_1|L_1')$ is closed under positive polynomial time tt-reducibility then there exists a pair (L_3, L_3') such that $\text{P}^{\exists \cdot \text{Leaf}_{\text{u}}^{\text{p}}(L_1|L_1')} = \text{Leaf}_{\text{u}}^{\text{p}}(L_3|L_3')$. If $\text{Leaf}_{\text{u}}^{\text{p}}(L_1|L_1')$ is complementary then $\text{Leaf}_{\text{u}}^{\text{p}}(L_3|L_3')$ is complementary too.*

Problem. Which is the status of the families not mentioned in Theorem 10.1-3?

In [BCS92] it is shown that there exist pairs $(L_1|L_1')$ and $(L_2|L_2')$ of leaf languages such that $\text{Leaf}^{\text{p}}(L_1|L_1') \cup \text{Leaf}^{\text{p}}(L_2|L_2')$ is not a leaf language class. Strong evidence for this fact is given by the leaf language classes NP and co-NP. If $\text{NP} \cup \text{co-NP}$ would be a leaf language class then $\text{NP} = \text{co-NP}$ [Bo04].

From Theorem 10 one obtains the following.

Corollary 1. *1. For every pair (L_1, L_1') of leaf languages and every class $\mathcal{K}$ of the modulo hierarchy over $\text{Leaf}^{\text{p}}(L_1|L_1')$ or the boolean hierarchy over $\text{Leaf}^{\text{p}}(L_1|L_1')$ there exists a pair (L_2, L_2') of leaf languages such that $\text{Leaf}^{\text{p}}(L_2|L_2') = \mathcal{K}$.*
2. *For every leaf language L_1 and every class $\mathcal{K}$ of the modulo hierarchy over $\text{Leaf}^{\text{p}}(L_1)$ there exists a leaf language L_2 such that $\text{Leaf}^{\text{p}}(L_2) = \mathcal{K}$.*
3. *For every paddable leaf language L_1 and every class $\mathcal{K}$ of the boolean hierarchy over $\text{Leaf}^{\text{p}}(L_1)$ there exists a leaf language L_2 such that $\text{Leaf}^{\text{p}}(L_2) = \mathcal{K}$.*
4. *For every $\mathcal{K} \in \{\text{NP}(k) : k \geq 1\} \cup \{\Sigma_k^{\text{p}} : k \geq 1\} \cup \{\Pi_k^{\text{p}} : k \geq 1\} \cup \{\Delta_k^{\text{p}} : k \geq 1\}$ and every class $\mathcal{K}$ from the modulo hierarchy over* P *there exists a regular, paddable language L such that $\text{Leaf}^{\text{p}}(L) = \mathcal{K}$.*

5 Regular Leaf Languages

Which classes can be defined by regular leaf languages? Which is the largest such class? The latter questions is answered by the following theorem. Let A_5 be the group of even permutations of $(1,2,3,4,5)$, and let a_1 be the identical permutation of $(1,2,3,4,5)$. Obviously, the language

$$M_5 =_{\mathrm{def}} \{p_1 p_2 \ldots p_m : m \geq 1 \wedge p_1, p_2, \ldots, p_m \in \mathrm{A}_5 \wedge p_1 \cdot p_2 \cdot \ldots \cdot p_m = a_1\}$$

is regular. Let REG denote the class of all regular languages.

Theorem 11. [HLSVW93] $\mathrm{Leaf}^{\mathrm{p}}(M_5) = \mathrm{Leaf}^{\mathrm{p}}(\mathrm{REG}) = \mathrm{PSPACE}$.

Which regular leaf languages describe the class PSPACE? This can be answered by looking at certain algebraic properties of their syntactic monoids. (For a survey on the connections between languages, monoids, and finite model theory see [Pi96].) Notice that it is a widely believed conjecture that MOD-PH is a proper subclass of PSPACE. Hence Statement 1 and Statement 2 of the following theorem have a dichotomic character.

Theorem 12. [HLSVW93]

1. *If the syntactic monoid of L is not solvable then* $\mathrm{Leaf}^{\mathrm{p}}(L) = \mathrm{PSPACE}$.
2. *If the syntactic monoid of L is solvable then* $\mathrm{Leaf}^{\mathrm{p}}(L) \subseteq \mathrm{MOD\text{-}PH}$.
3. *If the syntactic monoid of L is aperiodic then* $\mathrm{Leaf}^{\mathrm{p}}(L) \subseteq \mathrm{PH}$.

There is an interesting correspondence between classes of languages with an aperiodic syntactic monoid and the classes of the polynomial time hierarchy. The syntactic monoid of L is aperiodic if and only if L is starfree (Schützenberger 1965). The class SF of starfree languages is defined as the smallest class of languages which containes all finite languages and which is closed under union, complementation, and concatenation. Two so-called *dot-depth hierarchies* within the class of starfree languages have been studied intensively. Roughly speaking, the levels of these hierarchies are defined by the number of alternations between concatenations and boolean set operations in the generation of a starfree language. For a class $\mathcal{K}$, let $\mathrm{BC}(\mathcal{K})$ be the *boolean closure* of $\mathcal{K}$, i.e., the closure of $\mathcal{K}$ under union and complementation, and let $\mathrm{Pol}(\mathcal{K})$ be the closure of $\mathcal{K}$ under union and concatenation. The *Cohen-Brzozowski hierarchy* (Cohen and Brzozowski 1971) consists of the classes $\mathcal{B}_{k/2}$ for $k \geq 0$ and the *Straubing-Thérien hierarchy* (Straubing and Thérien 1981) consists of the classes $\mathcal{L}_{k/2}$ for $k \geq 1$. These classes are defined by

$$\begin{array}{llll}
\mathcal{B}_{k/2} & =_{\mathrm{def}} \bigcup_{A \text{ finite alphabet}} \mathcal{B}^A_{k/2} & \mathcal{L}_{k/2} & =_{\mathrm{def}} \bigcup_{A \text{ finite alphabet}} \mathcal{L}^A_{k/2} \\
\mathcal{B}^A_0 & =_{\mathrm{def}} \{\{w\} : w \in A^*\} \cup & & \\
 & \qquad \cup \{wA^*v : w, v \in A^*\} & & \\
\mathcal{B}^A_{1/2} & =_{\mathrm{def}} \mathrm{Pol}(\{\{w\} : w \in A^*\} \cup \{A^*\}) & \mathcal{L}^A_{1/2} & =_{\mathrm{def}} \mathrm{Pol}(\{A^* a A^* : a \in A\}) \\
\mathcal{B}^A_k & =_{\mathrm{def}} \mathrm{BC}(\mathcal{B}_{k-1/2}) & \mathcal{L}^A_k & =_{\mathrm{def}} \mathrm{BC}(\mathcal{L}_{k-1/2}) \\
\mathcal{B}^A_{k+1/2} & =_{\mathrm{def}} \mathrm{Pol}(\mathcal{B}_k) & \mathcal{L}^A_{k+1/2} & =_{\mathrm{def}} \mathrm{Pol}(\mathcal{L}_k)
\end{array}$$

It is well known that $\mathcal{L}_{k-1/2} \subset \mathcal{B}_{k-1/2} \subset \mathcal{L}_{k+1/2}$ and $\mathcal{L}_k \subset \mathcal{B}_k \subset \mathcal{L}_{k+1}$ for $k \geq 1$, and hence $\bigcup_{k \geq 1} \mathcal{L}_{k/2} = \bigcup_{k \geq 1} \mathcal{B}_{k/2} = \mathrm{SF}$. Defining $L_1 =_{\mathrm{def}} 0^*1(0 \cup 1)^*$

and $L_k =_{\mathrm{def}} \{0,1,\ldots k\}^*k(\{0,1,\ldots,k-1\}^* \setminus L_{k-1})k\{0,1,\ldots k\}^*$ for $k \geq 2$ one obtains $L_k \in \mathcal{L}_{k-1/2}$, and in [SW04] it is shown that $L_k \notin \mathcal{B}_{k-1}$ for $k \geq 1$.

In the leaf language concept the levels of the dot-depth hierarchies correspond exactly to the levels of the polynomial time hierarchy.

Theorem 13. *1.* [HLSVW93] $\mathrm{Leaf}^{\mathrm{p}}(\mathrm{SF}) = \mathrm{PH}$.
2. [HLSVW93] $\mathrm{Leaf}^{\mathrm{p}}(\mathcal{B}_0) = \mathrm{P}$.
3. [BV98] $\mathrm{Leaf}^{\mathrm{p}}(L_k) = \mathrm{Leaf}^{\mathrm{p}}(\mathcal{L}_{k-1/2}) = \mathrm{Leaf}^{\mathrm{p}}(\mathcal{B}_{k-1/2}) = \Sigma_k^{\mathrm{p}}$ *for* $k \geq 1$.
4. [HLSVW93] $\mathrm{Leaf}^{\mathrm{p}}(\mathcal{L}_k) = \mathrm{Leaf}^{\mathrm{p}}(\mathcal{B}_k) = \mathrm{BC}(\Sigma_k^{\mathrm{p}})$ *for* $k \geq 1$.
5. [BSS99, BLSTT04] $\mathrm{Leaf}^{\mathrm{p}}(\mathcal{L}_{k+1/2} \cap \text{co-}\mathcal{L}_{k+1/2}) = \mathrm{Leaf}^{\mathrm{p}}(\mathcal{B}_{k+1/2} \cap \text{co-}\mathcal{B}_{k+1/2}) = \Delta_{k+1}^{\mathrm{p}}$ *for* $k \geq 0$.

In [Sch00] it is shown that $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(\mathrm{RTL}) = \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(\mathrm{RPTL}) = \Delta_2^{\mathrm{p}}$, where PLT and RPLT are proper subclasses of $\mathcal{B}_{3/2} \cap \text{co-}\mathcal{B}_{3/2}$ defined by restricted temporal logic.

The correspondence between the dot-depth hierarchies on the one side and the polynomial time hierarchy on the other side is even stronger than indicated by the previous theorem. It applies also to the boolean hierachies over the various levels of the polynomial time hierarchy.

Theorem 14. *1.* [SW98, BKS99] $\mathrm{Leaf}^{\mathrm{p}}(\mathcal{L}_{1/2}(m)) = \mathrm{Leaf}^{\mathrm{p}}(\mathcal{B}_{1/2}(m)) = \mathrm{NP}(m)$ *for* $m \geq 1$.
2. [Se01] $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(\mathcal{B}_{k-1/2}(m)) = \Sigma_k^{\mathrm{p}}(m)$ *for* $k, m \geq 1$.

It is also shown in [Se01] that some refinements of the $\mathcal{B}_{m-1/2}(k)$ hierarchy can be related to the corresponding refinements of the $\Sigma_m^{\mathrm{p}}(k)$ hierarchy in the same way as in Theorem 14.

We have seen that P is the minimal class which can be defined by leaf languages. We do also know the "next larger" classes which can be defined by regular leaf languages. Let us start with the unbalanced case.

Theorem 15. *Let L be a regular language.*
1. [Bo94a] $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L) = \mathrm{P}$ *or* $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L) \supseteq \mathrm{NP}$ *or* $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L) \supseteq \text{co-NP}$ *or* $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L) \supseteq \mathrm{Mod}_p\mathrm{P}$ *for some prime number* p.
2. [BKS99] *If* $L \in \mathcal{B}_0$ *then* $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L) = \mathrm{P}$.
3. [BKS99] *If* $L \in \mathcal{B}_{1/2} \setminus \mathcal{B}_0$ *then* $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L) = \mathrm{NP}$.
4. [BKS99] *If* $L \in \text{co-}\mathcal{B}_{1/2} \setminus \mathcal{B}_0$ *then* $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L) = \text{co-NP}$.
5. [BKS99] *If* $L \in \mathrm{SF} \setminus \mathcal{B}_{1/2}$ *then* $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L) \supseteq \forall\mathrm{P} = \text{co-NP}$ *or* $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L) \supseteq \text{co-}\exists!\cdot\mathrm{P}$.
6. [BLSTT04] *If* $L \in \mathcal{L}_{3/2} \cap \text{co-}\mathcal{L}_{3/2}$ *then* $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L) \subseteq \mathrm{BC(NP)}$ *or* $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L) = \Delta_2^{\mathrm{p}}$.
7. [Sch00] *If* $L \in \mathrm{SF} \setminus \mathcal{B}_{3/2}$ *then* $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L) \supseteq \forall\cdot\mathrm{UP}$ *or* $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L) \supseteq \text{co-}\exists!\cdot\mathrm{UP}$.

Let us hint at the similar structure of Statement 5 and Statement 7 which are proved using *forbidden pattern* characterizations of the classes $\mathcal{B}_{1/2}$ and $\mathcal{B}_{3/2}$. Similar results for higher classes $\mathcal{B}_{k/2}$ could be obtained from forbidden pattern characterizations of these classes which are unfortunately not known so far.

For the proper subclasses PLT and RPLT of $\mathcal{B}_{3/2} \cap \text{co-}\mathcal{B}_{3/2}$ (see the remark after Theorem 13) the following is shown in [Sch00]. If $L \notin \text{RTL} \cap \text{RPTL}$ then $\text{Leaf}_{\text{u}}^{\text{p}}(L) \supseteq \Delta_2^{\text{p}}$ or $\text{Leaf}_{\text{u}}^{\text{p}}(L) \supseteq \text{Leaf}_{\text{u}}^{\text{p}}((0^*10^*2)^*0^*)$ or $\text{Leaf}_{\text{u}}^{\text{p}}(L) \supseteq \text{co-Leaf}_{\text{u}}^{\text{p}}((0^*10^*2)^*0^*)$ or $\text{Leaf}_{\text{u}}^{\text{p}}(L) \supseteq \text{Mod}_p\text{P}$ for some prime number p.

Under the assumption that the polynomial time hierarchy does not collapse one can conclude from Theorem 15 characterizations of those regular leaf languages which define the classes P, NP and co-NP, resp.

Corollary 2. [BKS99] *Assume that the polynomial time hierarchy does not collapse. Let L be a regular language.*

1. $\text{Leaf}_{\text{u}}^{\text{p}}(L) = \text{P} \Leftrightarrow L \in \mathcal{B}_0$.
2. $\text{Leaf}_{\text{u}}^{\text{p}}(L) = \text{NP} \Leftrightarrow L \in \mathcal{B}_{1/2} \setminus \mathcal{B}_0$.
3. $\text{Leaf}_{\text{u}}^{\text{p}}(L) = \text{co-NP} \Leftrightarrow L \in \text{co-}\mathcal{B}_{1/2} \setminus \mathcal{B}_0$.

Now we consider the balanced case. For a class $\mathcal{K}$ of languages, let $\mathcal{R}_{\text{m}}^{\text{plt}}(\mathcal{K})$ be the class of languages which are $\leq_{\text{m}}^{\text{plt}}$-reducible (see Section 7) to a language in $\mathcal{K}$.

Theorem 16. [Gl04] *Let L be a regular language.*

1. $\text{Leaf}_{\text{b}}^{\text{p}}(L) = \text{P}$ *or* $\text{Leaf}_{\text{b}}^{\text{p}}(L) \supseteq \text{NP}$ *or* $\text{Leaf}_{\text{b}}^{\text{p}}(L) \supseteq \text{co-NP}$ *or* $\text{Leaf}_{\text{b}}^{\text{p}}(L) \supseteq \text{Mod}_p\text{P}$ *for some prime number p.*
2. *If $L \in \mathcal{R}_{\text{m}}^{\text{plt}}(\mathcal{B}_0)$ then* $\text{Leaf}_{\text{b}}^{\text{p}}(L) = \text{P}$.
3. *If $L \in \mathcal{R}_{\text{m}}^{\text{plt}}(\mathcal{B}_{1/2}) \setminus \mathcal{R}_{\text{m}}^{\text{plt}}(\mathcal{B}_0)$ then* $\text{Leaf}_{\text{b}}^{\text{p}}(L) = \text{NP}$.
4. *If $L \in \mathcal{R}_{\text{m}}^{\text{plt}}(\text{co-}\mathcal{B}_{1/2}) \setminus \mathcal{R}_{\text{m}}^{\text{plt}}(\mathcal{B}_0)$ then* $\text{Leaf}_{\text{b}}^{\text{p}}(L) = \text{co-NP}$.
5. *If $L \in \text{SF} \setminus \mathcal{R}_{\text{m}}^{\text{plt}}(\mathcal{B}_{1/2})$ then* $\text{Leaf}_{\text{b}}^{\text{p}}(L) \supseteq \text{co-NP}$ *or* $\text{Leaf}_{\text{b}}^{\text{p}}(L) \supseteq \text{co-1-NP}$.

Corollary 3. *Assume that the polynomial time hierarchy does not collapse. Let L be a regular language.*

1. $\text{Leaf}_{\text{b}}^{\text{p}}(L) = \text{P} \Leftrightarrow L \in \mathcal{R}_{\text{m}}^{\text{plt}}(\mathcal{B}_0)$.
2. $\text{Leaf}_{\text{b}}^{\text{p}}(L) = \text{NP} \Leftrightarrow L \in \mathcal{R}_{\text{m}}^{\text{plt}}(\mathcal{B}_{1/2}) \setminus \mathcal{R}_{\text{m}}^{\text{plt}}(\mathcal{B}_0)$.
3. $\text{Leaf}_{\text{b}}^{\text{p}}(L) = \text{co-NP} \Leftrightarrow L \in \mathcal{R}_{\text{m}}^{\text{plt}}(\text{co-}\mathcal{B}_{1/2}) \setminus \mathcal{R}_{\text{m}}^{\text{plt}}(\mathcal{B}_0)$.

Problem. From Theorem 3, Corollary 2, and Corollary 3 we obtain for every regular paddable language L that $L \in \mathcal{R}_{\text{m}}^{\text{plt}}(\mathcal{B}_{k/2})$ implies $L \in \mathcal{B}_{k/2}$ for $k = 0, 1$. Is this true for every $k \geq 2$?

However, the classes $\mathcal{B}_0$ and $\mathcal{B}_{1/2}$ are not closed under $\leq_{\text{m}}^{\text{plt}}$-reducibility as shown in the following theorem.

Theorem 17.
1. [Wa01] *A regular language $L \in \Sigma^*$ is in $\mathcal{R}_{\text{m}}^{\text{plt}}(\mathcal{B}_0)$ if and only if there exists an $r \geq 0$ such that L is the finite union of sets $w(\Sigma^r)^*v$.*
2. [Gl04] $\text{REG} \cap \mathcal{R}_{\text{m}}^{\text{plt}}(\mathcal{B}_{1/2}) = \text{Pol}(\text{REG} \cap \mathcal{R}_{\text{m}}^{\text{plt}}(\mathcal{B}_0))$.

By Theorem 17.1 we obtain that the non-starfree language $(11)^*$ is in $\mathcal{R}_{\text{m}}^{\text{plt}}(\mathcal{B}_0)$. However, $\text{SF} \cap \mathcal{R}_{\text{m}}^{\text{plt}}(\mathcal{B}_0) = \mathcal{B}_0$ [Wa01]. The latter is not true for $\mathcal{R}_{\text{m}}^{\text{plt}}(\mathcal{B}_{1/2})$. It was proved in [Gl04] that, for every $k \geq 1$, there exists a starfree language $L_k \in \mathcal{R}_{\text{m}}^{\text{plt}}(\mathcal{B}_{1/2}) \setminus \mathcal{B}_k$. For these languages the balanced and the unbalanced leaf language classes could differ because $\text{Leaf}_{\text{b}}^{\text{p}}(L_k) = \text{NP}$ and

$\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_k) \supseteq \mathrm{U}\Sigma_k^{\mathrm{p}}$ where $\mathrm{U}\Sigma_k^{\mathrm{p}}$ is the k-the level of the UP-*hierarchy* defined by $\mathrm{U}\Sigma_1^{\mathrm{p}} =_{\mathrm{def}} \mathrm{UP}$ and $\mathrm{U}\Sigma_{k+1}^{\mathrm{p}} =_{\mathrm{def}} \mathrm{U}\cdot\text{co-}\mathrm{U}\Sigma_k^{\mathrm{p}}$ for $k \geq 1$. There are oracles relative to which $\mathrm{U}\Sigma_k^{\mathrm{p}} \not\subseteq \Sigma_{k-1}^{\mathrm{p}}$.

For a reducibility under which all classes of the Cohen-Brzozowski hierarchy are closed see [SW04].

In [Ga98] the classes $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L)$ and $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L)$ was investigated for regular languages L which can be accepted by finite deterministic automata with at most three states.

6 Other Leaf Languages

So far we have looked at classes which are defined by regular leaf languages. What happens if we choose leaf languages from other classes, in particular from given complexity classes? Since the leaf word $\beta_M(x)$ of a machine M can be exponentially longer than the input word x, one can expext a maximally exponential blowup of the complexity when going from a leaf language to the languages accepted with this leaf language. Indeed this exponential blowup does really happen. In particular, we have

Theorem 18. [HLSVW93] *Let* $s \geq \log$ *and* $t \geq \mathrm{id}$*.*

1. $\mathrm{Leaf}^{\mathrm{p}}(\mathrm{DSPACE}(s)) = \mathrm{DSPACE}(s \circ 2^{\mathrm{Pol}})$.
2. $\mathrm{Leaf}^{\mathrm{p}}(\mathrm{NSPACE}(s)) = \mathrm{NSPACE}(s \circ 2^{\mathrm{Pol}})$.
3. $\mathrm{Leaf}^{\mathrm{p}}(\mathrm{DTIME}(t)) = \mathrm{DTIME}(\mathrm{Pol} \circ t \circ 2^{\mathrm{Pol}})$.
4. $\mathrm{Leaf}^{\mathrm{p}}(\mathrm{NTIME}(t)) = \mathrm{NTIME}(\mathrm{Pol} \circ t \circ 2^{\mathrm{Pol}})$.

Corollary 4. *1.* $\mathrm{Leaf}^{\mathrm{p}}(\mathrm{PSPACE}) = \mathrm{DSPACE}(2^{\mathrm{Pol}})$.
2. $\mathrm{Leaf}^{\mathrm{p}}(\mathrm{NP}) = \mathrm{NTIME}(2^{\mathrm{Pol}})$.
3. $\mathrm{Leaf}^{\mathrm{p}}(\mathrm{P}) = \mathrm{DTIME}(2^{\mathrm{Pol}})$.
4. $\mathrm{Leaf}^{\mathrm{p}}(\mathrm{L}) = \mathrm{Leaf}^{\mathrm{p}}(\mathrm{NL}) = \mathrm{Leaf}^{\mathrm{p}}(\mathrm{DSPACE}(\log^k)) = \mathrm{PSPACE}$.

Because of Theorem 11 and Statement 4 of the preceding theorem we have also $\mathrm{Leaf}^{\mathrm{p}}(\mathrm{CFL}) = \mathrm{Leaf}^{\mathrm{p}}(\mathrm{DCFL}) = \mathrm{Leaf}^{\mathrm{p}}(\mathrm{NC}^1) = \mathrm{PSPACE}$.

The next theorem gives another example of the phenomenon that the balanced and the unbalanced leaf language classes for the same leaf languages can differ (unless the polynomial time hierarchy collapses). Note that $\mathcal{B}_{k-1/2} \subseteq \Sigma_k$-LOGTIME for $k \geq 0$ and $\mathrm{SF} \subseteq \mathrm{AC}^0$.

Theorem 19. [JMT94, HVW96]

1. $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(\Sigma_k\text{-LOGTIME}) = \Sigma_k^{\mathrm{p}}$ *and* $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(\Sigma_k\text{-LOGTIME}) = (\Sigma_k^{\mathrm{p}})^{\mathrm{PP}}$ *for* $k \geq 0$.
2. $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(\mathrm{AC}^0) = \mathrm{PH}$ *and* $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(\mathrm{AC}^0) = \mathrm{PH}^{\mathrm{PP}}$.

In [Vo98] classes $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L)$ were studied where L is accepted by special circuits.

7 Relativized Leaf Language Classes

In this section we will consider relativized leaf language classes, i.e., the classes of type $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L_1|L_2)^X$, $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_2)^X$, $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L)^X$, and $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L)^X$. It is worth mentioning that Theorem 4 remain valid if all leaf language classes there are attached with a fixed oracle X [Bo94b]. The next theorem is in a way an analogue to Theorem 5. From these theorems it follows that if a class $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_2)$ has a $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete set then the $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_2)^X$ has a $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete set for every oracle X.

Theorem 20. [BCS92] *A leaf language class* $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_2)^X$ *has a* $\leq_{\mathrm{m}}^{\mathrm{p}}$*-complete set for every oracle* X *if and only if there exists a language* L *such that* $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_2) = \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L)$.

The use of oracles is an important tool in complexity theory. Several central problems about the inclusion between complexity classes are unresolved, e.g., the famous problem of whether $\mathrm{P} = \mathrm{NP}$. However, it was possible to prove that there are oracles X, Y such that $\mathrm{P}^X \neq \mathrm{NP}^X$ and $\mathrm{P}^Y = \mathrm{NP}^Y$. That means that we cannot prove $\mathrm{P} = \mathrm{NP}$ or $\mathrm{P} \neq \mathrm{NP}$ by using relativizable methods, i.e., methods which work in the same way when using additionally an oracle set. We will present here a relatively easy to handle criterion for the fact that there are oracles X such that $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_1')^X \not\subseteq \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_2|L_2')^X$ and $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L_1|L_1')^X \not\subseteq \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L_2|L_2')^X$.

To get a criterion for $\forall X(\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_1')^X \subseteq \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_2|L_2')^X)$ we define the *polylogtime reducibility* $\leq_{\mathrm{m}}^{\mathrm{plt}}$. A function $g : \Sigma_1^* \to \Sigma_2^*$ is *polylogtime computable* if there exist a $k \geq 1$ such that g is computed in $\mathcal{O}(\log^k n)$ time by a Turing machine which accesses the input as an oracle. For pairs (L_1, L_1') and (L_2, L_2') of leaf languages let

$$
\begin{array}{rl}
(L_1|L_1') \leq_{\mathrm{m}}^{\mathrm{plt}} (L_2|L_2') \Leftrightarrow_{\mathrm{def}} & \text{there exist polylogtime computable functions} \\
& g, h \text{ such that} \\
& x \in L_1 \to g(x,1)g(x,2)\ldots g(x,h(x)) \in L_2 \text{ and} \\
& x \in L_1' \to g(x,1)g(x,2)\ldots g(x,h(x)) \in L_2'
\end{array}
$$

Instead of $(L_1|\overline{L_1}) \leq_{\mathrm{m}}^{\mathrm{plt}} (L_2|\overline{L_2})$ we also write $L_1 \leq_{\mathrm{m}}^{\mathrm{plt}} L_2$. For example, $0^*10^*1(0 \cup 1)^* \leq_{\mathrm{m}}^{\mathrm{plt}} 0^*1(0 \cup 1)^*$ but $1^* \not\leq_{\mathrm{m}}^{\mathrm{plt}} 0^*1(0 \cup 1)^*$ (1^* taken here as a subset of $\{0,1\}^*$).

Theorem 21. [BCS92, Ve93] *For all pairs* (L_1, L_1') *and* (L_2, L_2') *there holds* $(L_1, L_1') \leq_{\mathrm{m}}^{\mathrm{plt}} (L_2, L_2')$ *if and only if* $\forall X(\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_1')^X \subseteq \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_2|L_2')^X)$.
In particular, $L_1 \leq_{\mathrm{m}}^{\mathrm{plt}} L_2$ *if and only if* $\forall X(\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1)^X \subseteq \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_2)^X)$.

Corollary 5. *If* $(L_1, L_1') \not\leq_{\mathrm{m}}^{\mathrm{plt}} (L_2, L_2')$ *then there exists an oracle* X *such that* $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_1')^X \not\subseteq \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_2|L_2')^X$.

The preceding corollary can be used very elegantly to show that there are relativized worlds in which some inclusion between complexity classes does not hold. For example, to prove that there exists an oracle X such that $\text{co-NP}^X \not\subseteq$

NP^X (and hence $\mathrm{P}^X \neq \mathrm{NP}^X$) it is sufficient to prove $1^* \not\leq_{\mathrm{m}}^{\mathrm{plt}} 0^*1(0 \cup 1)^*$ (1^* taken here as a subset of $\{0,1\}^*$). To do that assume that there exists such a reduction. Hence a word from 1^n is mapped to a word $0^{m-1}1w$ by the reducing function. Because this function is polylogtime computable, not every symbol of the input 1^n can be scanned when computing the m-th symbol of $0^{m-1}1w$ (if n is sufficiently large). Change a not scanned symbol of 1^n from 1 to 0. This results in word not in 1^*, but the reducing function produces nevertheless a word from $0^*1(0 \cup 1)^*$, a contradiction.

A language L is called *commutative* if the membership of a word x to L depends only on the number of occurences of every alphabet symbol in x and not on the order in which the symbols occur in x. For the special case of commutative, paddable leaf languages the criterion $L_1 \leq_{\mathrm{m}}^{\mathrm{plt}} L_2$ for $\forall X(\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1)^X \subseteq \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_2)^X)$ can be formulated in a much more easy and applicable way. A commutative, paddable language $L \subseteq \{0,1,\ldots k\}^*$ with "padding symbol" 0 is determined by the set $V_L =_{\mathrm{def}} \{(n_1, n_2, \ldots, n_k) : 1^{n_1}2^{n_2}\ldots k^{n_k} \in L\}$ of *Parikh vectors.* Such a language L is said to be *of bounded significance* if there exists an $m \in \mathbb{N}$ such that $(n_1, n_2, \ldots, n_k) \in V_L \leftrightarrow (\min(n_1, m), \min(n_2, m), \ldots, \min(n_k, m)) \in V_L$ for all $n_1, n_2, \ldots, n_k \in \mathbb{N}$. For $u, v \in \mathbb{N}^k$, we define the *multinomial coefficient* $\binom{u}{v} =_{\mathrm{def}} \prod_{i=1}^{k} \binom{u_i}{v_i}$.

Theorem 22. [CHVW98] *For commutative, paddable language of bounded significance $L_1 \subseteq \{0,1,\ldots k\}^*$ and $L_2 \subseteq \{0,1,\ldots m\}^*$ the following are equivalent:*

(1) $\forall X(\mathrm{Leaf}^{\mathrm{p}}(L_1)^X \subseteq \mathrm{Leaf}^{\mathrm{p}}(L_2)^X)$
(2) $L_1 \leq_{\mathrm{m}}^{\mathrm{plt}} L_2$
(3) there exist $p_1, \ldots, p_m : \mathbb{N}^k \rightarrow \mathbb{N}$ which are positive linear combinations of multinomial coefficients such that $v \in V_{L_1} \leftrightarrow (p_1(v), \ldots, p_m(v)) \in V_{L_2}$.

For a commutative, paddable language $L \subseteq \{0,1,\ldots k\}^*$ define $\mathrm{m}^+(L)$ ($\mathrm{m}^-(L)$) to be the maximum natural number r such that there exist $v_1, v_2, \ldots, v_r$ such that $v_1 \leq v_2 \leq \cdots \leq v_r$, $v_1 \in V_L$ ($v_1 \notin V_L$), and $v_i \in V_L \leftrightarrow v_{i+1} \notin V_L$ for $i = 1, 2, \ldots, k-1$. If there is not such a maximum natural number then $\mathrm{m}^+(L) =_{\mathrm{def}} \mathrm{m}^-(L) =_{\mathrm{def}} \infty$.

Theorem 23. [CHVW98] *Let L_1 and L_2 be commutative, paddable languages, and let L_2 be of bounded significance. If $\forall X(\mathrm{Leaf}^{\mathrm{p}}(L_1)^X \subseteq \mathrm{Leaf}^{\mathrm{p}}(L_2)^X)$ then $\mathrm{m}^+(L_1) \leq \mathrm{m}^+(L_2)$ and $\mathrm{m}^-(L_1) \leq \mathrm{m}^-(L_2)$.*

Theorem 24. [CHVW98] *For a commutative, paddable language L there holds $\forall X(\mathrm{Leaf}^{\mathrm{p}}(L)^X \subseteq \mathrm{NP}(k)^X)$ if and only if $\mathrm{m}^+(L) \leq k$.*

A similar criterion for $\forall X(\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L)^X \supseteq \mathrm{NP}(k)^X)$ is also proved in [CHVW98].

In [GKV03] a modification of Theorem 21 was proved that concerns *generic* oracles. We do not give a formal definition of genericity, but, informally speaking, a generic oracle is a typical oracle in the sense that it has all the properties that can be enforced by a stage construction (the main technique to construct oracles with certain desired properties). A generic oracle which separates e.g. P and NP also separates all other complexity classes which are separated by an oracle which is built by a stage construction. For arithmetic sets L_1 and L_2 the following three statements are equivalent: (1) $L_1 \leq_{\mathrm{m}}^{\mathrm{plt}} L_2$, (2) $(\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1)^X \subseteq \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_2)^X)$ for every generic oracle X, and (3) $(\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1)^X \subseteq \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_2)^X)$ for some generic oracle X.

Corollary 5 shows that if $(L_1, L_1') \not\leq_{\mathrm{m}}^{\mathrm{plt}} (L_2, L_2')$ then there exists an oracle X such that $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_1')^X \not\subseteq \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_2|L_2')^X$. In [BCS92] an additional condition to (L_1, L_1') is given such that if $L_1 \not\leq_{\mathrm{m}}^{\mathrm{plt}} L_2$ then there exists an oracle X such that $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_2|L_2')^X$ is even strongly separated from $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_1')^X$, i.e., there exists an infinite set in $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_2|L_2')^X$ which has no infinite subset in $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_1')^X$.

In [BCS91] conditions are given under which $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_1') \subseteq \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_2|L_2')$ implies $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_1')^X \subseteq \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_2|L_2')^X$ for every sparse oracle X.

In [Ve93] a polylogtime Turing reducibility $\leq_{\mathrm{T}}^{\mathrm{plt}}$ is defined in a natural way, and it is shown that $(L_1, L_1') \leq_{\mathrm{T}}^{\mathrm{plt}} (L_2, L_2')$ if and only if $\forall X(\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_1')^X \subseteq \mathrm{P}^{\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_2|L_2')^X})$

To get a similar criterion like the one in Theorem 21 for the unbalanced leaf language model, i.e., a criterion for $\forall X(\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L_1|L_2)^X \subseteq \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L_1'|L_2')^X)$ a new reducibilty is necessary: the *polynomial time tree reducibility* $\leq_{\mathrm{m}}^{\mathrm{ptt}}$. It is based on paths in finite trees rather than bits of words. Let Σ be a finite alphabet. We call a triple $t = (T, h, m)$ a *Σ-tree* if $T \subseteq \{0,1\}^*$ is finite (the set of paths), $h : T \to \Sigma$, and $m \in \mathbb{N}$ such that $\forall z \forall u((u$ is an initial word of $z \wedge z \in T) \to u \in T)$ and $\forall z(z \in T \to |z| \leq m)$. Let T_Σ be the set of all Σ-trees. The *leaf word* of a Σ-tree $t = (T, h, m)$ is defined as $\beta(t) =_{\mathrm{def}} h(r_1)h(r_2)\ldots h(r_k)$ where $r_1, r_2, \ldots, r_k \in \{0,1\}^*$ are the maximal paths of t in lexicographical order.

A function $f : T_{\Sigma_1} \to T_{\Sigma_2}$ is called a *polynomial time tree function* (for short: ptt function) if and only if there exist functions $g_1 : T_{\Sigma_1} \times \{0,1\}^* \times \mathbb{N} \to \{0,1\}$, $g_2 : T_{\Sigma_1} \times \{0,1\}^* \times \mathbb{N} \to \Sigma_2$, and a $k > 0$ such that

- g_1, g_2 are polynomial time computable in the length of the $\{0,1\}^*$ and $\mathbb{N}$ arguments where a tree $t \in T_{\Sigma_1}$ is accessed as an oracle, and
- $f(t) = (T', h', m^k)$ where $T' =_{\mathrm{def}} \{z : g_1^t(z, m) = 1\}$ and $h'(z) =_{\mathrm{def}} g_2^t(z, m)$ for every tree $t = (T, h, m) \in T_{\Sigma_1}$.

For $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$, we define $L_1 \leq_{\mathrm{m}}^{\mathrm{ptt}} L_2$ (L_1 is *ptt-reducible* to L_2) if there exists a ptt function $f : T_{\Sigma_1} \to T_{\Sigma_2}$ such that $\beta(t) \in L_1 \leftrightarrow \beta(f(t)) \in L_2$ for all $t \in T_{\Sigma_1}$. Instead of $(L_1|\overline{L_1}) \leq_{\mathrm{m}}^{\mathrm{ptt}} (L_2|\overline{L_2})$ we also write $L_1 \leq_{\mathrm{m}}^{\mathrm{ptt}} L_2$.

As examples, we have $(0^*10^*1)^*0^* \leq_{\mathrm{m}}^{\mathrm{ptt}} (11)^*$ and $(11)^* \not\leq_{\mathrm{m}}^{\mathrm{ptt}} 1$ whereas $(0^*10^*1)^*0^* \not\leq_{\mathrm{m}}^{\mathrm{plt}} (11)^*$ and $(11)^* \leq_{\mathrm{m}}^{\mathrm{plt}} 1$.

Theorem 25. [Wa04] *For pairs (L_1, L_1') and (L_2, L_2') of leaf languages there holds $(L_1, L_1') \leq_{\mathrm{m}}^{\mathrm{ptt}} (L_2, L_2')$ if and only if $\forall X(\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L_1|L_1')^X \subseteq \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L_2|L_2')^X)$. In particular, $L_1 \leq_{\mathrm{m}}^{\mathrm{ptt}} L_2$ if and only if $\forall X(\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L_1)^X \subseteq \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L_2)^X)$.*

Corollary 6. *If $L_1 \not\leq_{\mathrm{m}}^{\mathrm{ptt}} L_2$ then there exists an oracle X such that $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L_1)^X \not\subseteq \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L_2)^X$.*

In [Wa04] a similar theorem was proved also for the leaf language model where the machines have unbalanced computation trees with ε-leaves.

8 Other Leaf Language Models

In [JMT94] leaf language models are considered which rest on logarithmic space and logarithmic time machines rather than polynomial time machines. We will present her some of their results about the logspace leaf language model. In the same systematics as for polynomial time machines we denote also the leaf language classes for logspace machines but we use an upper script log. In fact, we will need only classes $\mathrm{Leaf}^{\log}(L)$ because the following statements are valid for unbalanced as well as the balanced case.

Theorem 26. [JMT94]

1. $\mathrm{Leaf}^{\log}(\mathrm{PSPACE}) = \mathrm{EXPSPACE}$.
2. $\mathrm{Leaf}^{\log}(\mathrm{NP}) = \mathrm{NEXPTIME}$.
3. $\mathrm{Leaf}^{\log}(\mathrm{P}) = \mathrm{EXPTIME}$.
4. $\mathrm{Leaf}^{\log}(\mathrm{L}) = \mathrm{Leaf}^{\log}(\mathrm{NL}) = \mathrm{Leaf}^{\log}(\mathrm{DSPACE}(\log^k)) = \mathrm{PSPACE}$.
5. $\mathrm{Leaf}^{\log}(\mathrm{CFL}) = \mathrm{Leaf}^{\log}(\mathrm{DCFL}) = \mathrm{Leaf}^{\log}(\mathrm{NC}^1) = \mathrm{PSPACE}$.
6. $\mathrm{Leaf}^{\log}(\Sigma_k\text{-LOGTIME}) = \Sigma_k^{\mathrm{p}}$ *for* $k \geq 1$.
7. $\mathrm{Leaf}^{\log}(\mathrm{AC}^0) = \mathrm{PH}$.
8. $\mathrm{Leaf}^{\log}(\mathrm{REG}) = \mathrm{P}$.
9. $\mathrm{Leaf}^{\log}(\mathrm{SF}) = \mathrm{Leaf}^{\log}(\mathcal{B}_{k/2}) = \mathrm{NL}$ *for* $k \geq 1$.
10. $\mathrm{Leaf}^{\log}(\mathcal{B}_0) = \mathrm{L}$.

Interestingly, the results for the polynomial time leaf language model and the logspace leaf language model differ only for small classes. In particular the results for regular leaf languages are clearly different (unless P = PSPACE).

As in the polynomial time case the minimal leaf language class in the logspace model is L. Also the "next larger" such leaf languages classes are known.

Theorem 27. [Bo94a] *For a regular language L there holds* $\mathrm{Leaf}_{\mathrm{u}}^{\log}(L) = \mathrm{L}$ *or* $\mathrm{Leaf}_{\mathrm{u}}^{\log}(L) \supseteq \mathrm{NL}$ *or* $\mathrm{Leaf}_{\mathrm{u}}^{\log}(L) \supseteq \mathrm{Mod}_p\mathrm{L}$ *for some prime number p.*

In [CMTV98] a leaf language model on the base of NC^1 computations was considered with leaf language classes $\mathrm{Leaf}^{\mathrm{NC}^1}(L)$. The results are similar to those for the logspace model. Simple context-free languages L were found for which still $\mathrm{Leaf}^{\mathrm{NC}^1}(L) = \mathrm{PSPACE}$ holds true.

In [GV03] a leaf language model to compute functions was considered. Let D be the set of all (3×3)-matrices with entries from $\{-1,0,1\}$. For a string $x \in \mathrm{D}^*$ let $\langle x\rangle$ be the result of multiplying the matrices in x in the given order. The following result is the function analog of the *bottleneck* characterization of PSPACE (Cai und Furst 1991).

Theorem 28. [GV03] *A function f is computable in polynomial space if and only if there exist a nondeterministic polynomial time machine M that computes on every computation path a matrix from D such that*

$$f(x) = (1\,0\,0)\cdot\langle\beta_M(x)\rangle\cdot\begin{pmatrix}1\\0\\0\end{pmatrix}.$$

A general leaf language model for computing functions was defined and investigated in [KSV00].

In [Ko00] a leaf language model to describe classes of finite partitions was considered.

9 Leaf Languages for Recursion-Theoretic Classes

Is it possible to define recursion-theoretic classes with simple leaf languages? Since we understand the polynomial time hierarchy as the complexity-theoretic analogue of the arithmetic hierarchy we look for a leaf language model in which, analogously to Theorem 13, the classes of the Cohen-Brzozowski hierarchy generate the corresponding classes of the arithmetic hierarchy. We restrict ourselves to the case of balanced and complementary leaf language classes.

To reflect the quantifier structure in the arithmetic hierarchy where the quantifiers range over an infinite domain, leaf languages with finite objects like words are not suitable. We use infinite sequences from Σ^{ω^m} with $m \geq 1$, i.e., infinite sequences of ordinality ω^m over the finite alphabet Σ. Such sequences can be represented by total functions $\mathbb{N}^m \to \Sigma$. We generalize the leaf language concept to ω^m-languages. For $B \in \Sigma_1^*$ and $L \in \Sigma_2^{\omega^m}$ let

$$B \in \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{rec}}(L) \Leftrightarrow_{\mathrm{def}} \text{there exists a computable total } g : \Sigma_1^* \times \mathbb{N}^m \to \Sigma_2 \text{ such that } x \in B \leftrightarrow g(x) \in L \text{ for all } x \in \Sigma_1^*$$

where $g(x)(i_1, \dots, i_m) =_{\mathrm{def}} g(x, i_1, \dots, i_m)$ for $x \in \Sigma_1^*$ and $i_1, \dots, i_m \in \mathbb{N}$.

Regular ω^m-languages ($m \geq 1$) are well studied (Büchi 1965, Choueka 1978, Wojciechowski 1985, Bedon and Carton 1998). One possible definition is the following: An ω^m-language is regular if it can be described by a monadic second order formula with the relations $<$, s (successor relation), min and max. According to the corresponding results on starfree languages of words (Thomas 1982) we define: An ω^m-language is starfree if it can be described by a monadic first order formula with the same relations. An ω^m-language is in the class $\mathcal{B}_{k-1/2}^{\omega,m}$ if it can be described by a monadic first order formula with the same relations whose quantifiers have a Σ_k-structure. Further, let $\mathcal{B}_k^{\omega,m} =_{\mathrm{def}} \mathrm{BC}(\mathcal{B}_{k-1/2}^{\omega,m})$ and $\mathcal{B}_{k/2}^{\omega} =_{\mathrm{def}} \bigcup_{m \geq 1} \mathcal{B}_{k/2}^{\omega,m}$ for $k \geq 1$. Finally, for $k \geq 1$, let Σ_k^{ar} be the k-th class of the arithmetic hierarchy.

Theorem 29. *For $k \geq 1$, there holds $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{rec}}(\mathcal{B}_{k-1/2}^{\omega}) = \Sigma_k^{ar}$ and $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{rec}}(\mathcal{B}_k^{\omega}) = \mathrm{BC}(\Sigma_k^{ar})$.*

We conjecture that the preceding result is not valid for fixed m. For $m = 1$ it is definitively not valid because we can prove that $\bigcup_{k \geq 1} \mathcal{B}_k^{\omega,1} = \mathrm{BC}(\Sigma_2^{ar})$. Generally, we conjecture $\bigcup_{k \geq 1} \mathcal{B}_k^{\omega,m} = \mathrm{BC}(\Sigma_{2m}^{ar})$ for $m \geq 2$.

We say that the leaf set $L \in \Sigma^{\omega^m}$ characterizes the m-degree of the set B if $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{rec}}(L) = \{C : C \leq_{\mathrm{m}} B\}$. Which leaf sets characterize m-degrees? Which m-degrees can be characterized by leaf sets? It is not hard to see that, if one modifies the definition of $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{rec}}(L)$ by omitting "total" then every leaf set L

characterizes the m-degree of $R(L) =_{\mathrm{def}} \{c(r,x) : u(r,x) \in L\}$ where u is a universal function for the $(n+1)$-ary computable functions.

It would be very useful to have a reducibility $\leq^*$ between ω^n-sets such that

$$L_1 \leq^* L_2 \Leftrightarrow \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{rec}}(L_1) \subseteq \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{rec}}(L_2).$$

A first idea is as follows. For $L_1 \subseteq \Sigma_1^{\omega^m}$ and $L_2 \subseteq \Sigma_2^{\omega^n}$ we define $L_1 \leq_{\mathrm{m}}^{\mathrm{rec}} L_2 \Leftrightarrow_{\mathrm{def}}$ there exists a total computable oracle function g such that $\xi \in L_1 \leftrightarrow g^\xi(0)g^\xi(1)g^\xi(2)\ldots \in L_2$ for all $\xi \in \Sigma_1^{\omega^m}$

Theorem 30. *If* $L_1 \leq_{\mathrm{m}}^{\mathrm{rec}} L_2$ *then* $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{rec}}(L_1) \subseteq \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{rec}}(L_2)$.

Problem. Does $L_1 \leq_{\mathrm{m}}^{\mathrm{rec}} L_2 \Leftrightarrow \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{rec}}(L_1) \subseteq \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{rec}}(L_2)$ hold true or do we need a weaker reducibility to achieve that?

10 Further General Models and the Leaf Language Model

In this section we will consider other general models to define complexity classes.

In [GP86] and [He92a] the model of *locally definable acceptance types* was introduced. Let $k \geq 2$, and let F be a finite set of functions over $\{1, 2, \ldots, k\}$. A nondeterministic polynomial time machine M is called an F-machine if it assigns to every node of indegree r of its computation tree on input x an r-ary function from F. The input is accepted if a circuit-like evaluation of the computation tree results in 1. The class $(F)P$ consists of all languages which can be accepted in this way by an F-machine. For every F there holds $\mathrm{P} \subseteq (F)P \subseteq \mathrm{PSPACE}$, and there are single finite functions f such that $(\{f\})P = \mathrm{PSPACE}$. The connection to the leaf language model is as follows. For every regular language L there is a binary associative function f such that $(\{f\})P = \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L)$, and for every binary associative function f there is a regular language L such that $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L) = \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L) = (\{f\})P$. More results about locally definable acceptance types can be found in [GP86, He92a, He92b, He94a, He94b, NR98, PV01]

In [BS01] the leaf language concept is generalized to operators, the *dot operators*. For a pair (L_1, L_2) of leaf languages over the alphabet $\{0,1\}$ the dot operator $(L_1, L_2)\cdot$ is defined by

$A \in (L_1, L_2) \cdot \mathcal{K} =_{\mathrm{def}}$ there exist a $B \in \mathcal{K}$ and a polynomial time computable function h such that for all x,
$x \in A \to \mathrm{c}_B(x,0)\mathrm{c}_B(x,1)\ldots\mathrm{c}_B(x,h(x)) \in L_1$ and
$x \notin A \to \mathrm{c}_B(x,0)\mathrm{c}_B(x,1)\ldots\mathrm{c}_B(x,h(x)) \in L_2$.

Furthermore, for a leaf language L define $L \cdot \mathcal{K} =_{\mathrm{def}} (L, \overline{L}) \cdot \mathcal{K}$. For example, $(0^*1(0\cup1)^*)\cdot\mathcal{K} = \exists\cdot\mathcal{K}$ for every class $\mathcal{K}$ of languages. The observation $(L_1, L_2)\cdot \mathrm{P} = \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_2)$ makes clear that the dot operator $(L_1, L_2)\cdot$ is a generalization of the leaf language class $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1|L_2)$. It is proved that $\forall\mathcal{K}(L\cdot\mathcal{K} \subseteq (L_1, L_2)\cdot\mathcal{K})$ if and only if $(L, \overline{L})$ is polylogtime monotone projection reducible to (L_1, L_2). This can be considered as a generalization of Theorem 21. This model can also be used to characterize reducibility notions. For example, there exists a language L_T such that for all languages A and B there holds $A \leq_{\mathrm{T}}^{\mathrm{p}} B \Leftrightarrow A \in L_T \cdot \{C : C \leq_{\mathrm{m}}^{\mathrm{p}} B\}$. Many reducibilities can be characterized in such a way.

In [BV98] and [GV01] close connections between leaf language definable complexity classes and classes defined by certain *Lindström quantifiers* were established.

Acknowledgements. I am grateful to Bernd Borchert, Tübingen, for many valuable remarks on preliminary versions of this survey. Also I am thankful to Christian Glaßer and Stephen Travers, Würzburg, for the interesting discussions about this paper.

References

[Aa04] S. Aaronson. The complexity zoo. http://www.complexityzoo.com/

[BCS91] D.P. Bovet, P. Crescenzi, R. Silvestri. Daniel P. Bovet, Pierluigi Crescenzi, Riccardo Silvestri. Complexity classes and sparse oracles. Proc. 6th IEEE Structure in Complexity Theory Conference, 1991, 102–108. Journal of Computer and System Sciences 50, 1995, 382–390.

[BCS92] D.P. Bovet, P. Crescenzi, R. Silvestri. A uniform approach to define complexity classes. *Theoret. Comp. Sci*, 104 (1992), 263–283.

[BDG88] J.L. Balcázar, J. Díaz and J. Gabarró. *Structural Complexity I*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1988.

[BDG90] J.L. Balcázar, J. Díaz and J. Gabarró. *Structural Complexity II*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1990.

[Bo94a] B. Borchert. On the Acceptance Power of Regular Languages. Proc. 11th Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Computer Science 775, Springer Verlag, 1994, pp. 533-542. Final version: Theoretical Computer Science 148, 1995, 207–225.

[Bo94b] B. Borchert. Predicate Classes, Promise Classes, and the Acceptance Power of Regular Languages. Ph.D. thesis, Universität Heidelberg, 1994.

[Bo04] B. Borchert. Personal communication.

[BKS99] B. Borchert, D. Kuske and F. Stephan. On existentially first–order definable languages and their relation to *NP. Theor. Informatics Appl.*, 33 (1999), 259–269.

[BLSTT04] B. Borchert, K-J. Lange, F. Stephan, P. Tesson, D. Thérien. The dot-depth and the polynomial hierarchy correspond on the delta levels. Technical Report 2004-03, Wilhelm-Schickard-Institut, Universität Tübingen. To appear in the proceedings of DLT 2004.

[BL96] B. Borchert, A. Lozano. Succinct Circuit Representations and Leaf Language Classes are Basically the Same Concept. Information Processing Letters 58, 1996, 211–215.

[BS97] B. Borchert, R. Silvestri. A Characterization of the Leaf Language Classes. Information Processing Letters 63, 1997, 153–158. Preliminary version: B. Borchert. Predicate classes and promise classes, Proc. 9th Structure in Complexity Theory Conference, 1994, 235–241.

[BS01] B. Borchert, R. Silvestri. Dot Operators. Theoretical Computer Science 262(1), 2001, 501–523.

[BV98] H.-J. Burtschick and H. Vollmer. Lindström Quatifiers and Leaf Language Definability. *Int. J. of Foundations of Computer Science*, 9 (1998), 277–294.

[BSS99] B. Borchert, H. Schmitz, F. Stephan. Unpublished manuscript, 1999.

[CHVW98] K. Cronauer, U. Hertrampf, H. Vollmer and K.W. Wagner. The chain method to separate counting classes. *Theory Comput. Systems*, 31 (1998), 93–108.

[CMTV98] H. Caussinus, P. McKenzie, D. Thérien, H. Vollmer. Nondeterministic NC1 Computation. Journal of Computer and System Sciences 57, 1998, 200–212.

[Ga98] M. Galota, Blattsprachen und endliche Automaten. Technical Report No. 205, University of Würzburg, Department of Computer Science, 1.

[GKV03] M. Galota, S. Kosub, H. Vollmer. Generic Separations and Leaf Languages. to appear in Mathematical Logic Quarterly, 2003.

[Gl04] Ch. Glaßer. Counting with Counterfree Automata Electronic Colloquium on Computational Complexity, Report TR04-011, 2004. Technical Report No. 315, University of Würzburg, 2004.

[GP86] L.M. Goldschlager, I. Parberry. On the Construction of Parallel Computers from Various Bases of Boolean Functions. Theoretical Computer Science 43, 1986, 43–58.

[GV01] M. Galota, H. Vollmer. A generalization of the Büchi-Elgot-Trakhtenbrot theorem. Proc. 15th Computer Science Logic 2001, Lecture Notes in Computer Science Vol. 2142, 2001, 355–368.

[GV03] M. Galota, H. Vollmer. Functions computable in polynomial space. Manuscript 2003.

[He92a] U. Hertrampf. Locally Definable Acceptance Types for Polynomial Time Machines. Proc. 9th Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Computer Science 577, Springer Verlag, 1992, 199–207.

[He92b] U. Hertrampf. Locally Definable Acceptance Types - the Three Valued Case. 1st Latin American Symposium in Theoretical Informatics, Lecture Notes in Computer Science 583, Springer Verlag, 1992, 262–271.

[He94a] U. Hertrampf. Complexity Classes with Finite Acceptance Types. Proc. 11th Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Computer Science 775, Springer Verlag, 1994, 543–553.

[He94b] U. Hertrampf. Complexity Classes Defined via k-Valued Functions. Proc. 9th Structure in Complexity Theory Conference, 1994, 224–234.

[HLSVW93] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer and K.W. Wagner. On the power of polynomial time bit-reductions, *Proc. 8th Structure in Complexity Theory*, 1993, 200–207.

[HVW96] U. Hertrampf, H. Vollmer and K.W. Wagner. On balanced vs. unbalanced computation trees, *Math. Systems Theory* 29 (1996), 411–421.

[JMT94] B. Jenner, P. McKenzie and D. Thérien. Logspace and logtime leaf languages, 9th Annual Conference on Structure in Complexity Theory 1994, 242–254. *Information and Computation*, 129 (1996), 21–33.

[Ko00] S. Kosub. On NP-partitions over posets with an application to reducing the set of solutions of NP problems. Proc. 25th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science Vol. 1893, 2000, 467–476.

[KSV00] S. Kosub, H. Schmitz, H. Vollmer. Uniformly defining complexity classes of functions. International Journal of Foundations of Computer Science 11(4), 2000, 525–551.

[NR98] R. Niedermeier, P. Rossmanith. Unambiguous computations and locally definable acceptance types. Theoretical Computer Science 194, 1998, 137–161.

[Pa94] Ch.H. Papadimitriou. Computational Complexity. Addison Wesley, 1994.

[PV01] T. Peichl, H. Vollmer. Finite automata with generalized acceptance criteria. Discrete Mathematics and Theoretical Computer Science 4, 2001, 179–192.

[Pi96] J.-E. Pin. Syntactic semigroups. In: G. Rozenberg and A. Salomaa (editors), Handbook of formal languages, Volume 1. 679–746. Springer, 1996.

[Sch00] H. Schmitz. The forbidden pattern approach to concatenation hierachies. Ph.D. Thesis, University of Würzburg 2000.

[SW98] H. Schmitz, K. Wagner. The Boolean Hierarchy over Level 1/2 of the Straubing-Therien Hierarchy. Technical Report No. 201, University of Würzburg, Department of Computer Science, 1998.

[Se01] V.L. Selivanov. Relating automata-theoretic hierarchies to complexity-theoretic hierarchies. Proc. 13th Conference on Fundamentals of Computation Theory 2001, Lecture Notes in Computer Science Vol. 2138, 2001, 323–334. Final version: Theoret. Informatics Appl. 36 (2002), 29–42.

[SW04] V.L. Selivanov, K.W. Wagner. A reducibility for the dot-depth hierarchy. Proc. of the 29th Intern. Symp. on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science Vol. 3153, 2004, 783–793.

[Tr02] St. Travers. Blattsprachen Komplexitätsklassen: Über Turing-Abschluss und Counting-Operatoren Studienarbeit, Universität Würzburg, Dezember 2002.

[Ve93] N.K. Vereshchagin. Relativizable and non-relativizable theorems in the polynomial theory of algorithms. *Izvestiya Rossiiskoi Akademii Nauk*, 57 (1993), 51–90 (in Russian).

[Ve98] H. Veith. Succinct Representation, Leaf Languages and Projection Reductions. Information and Computation 142 (1998), 207–236. Preliminary version: "Succinct Representation and Leaf Languages" in: Proc. 11th Annual IEEE Conference on Computational Complexity (CCC), 1996, 118–126.

[Vo98] H. Vollmer. Relating polynomial time to constant depth. Theoretial Computer Science 207 (1998), 159–170

[Vo99] H. Vollmer. Uniform characterizations of complexity classes. ACM SIGACT-Newsletter 30(1), 1999, 17–27.

[Vo03] H. Vollmer. Complexity theory made easy - the formal language approach to the definition of complexity classes. Proc. 7th Developments in Language Theory, 2003.

[Vo04] H. Vollmer. The Leaf Language Homepage.
http://www.thi.uni-hannover.de/forschung/leafl/

[Wa01] K.W. Wagner. A reducibility and complete sets for the dot-depth hierarchy. Manuscript.

[Wa04] K.W. Wagner. New BCSV theorems. Technical Report 337, Institut für Informatik, Universität Würzburg.

Computational Completeness of P Systems with Active Membranes and Two Polarizations

Artiom Alhazov[1], Rudolf Freund[2], and Gheorghe Păun[3]

[1] Research Group on Mathematical Linguistics, Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
artiome.alhazov@estudiants.urv.es
and
Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD 2028, Moldova
artiom@math.md

[2] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, A-1040 Wien, Austria
rudi@emcc.at

[3] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
and
Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es

Abstract. P systems with active membranes using only two electrical charges and only rules of type (a), i.e., evolution rules used in parallel in the regions of the membrane system, and of type (c), i.e., communication rules sending out an object of a membrane thereby possibly changing the polarization of this membrane, assigned to at most two membranes are shown to be computationally complete, which improves the previous result of this type with respect to the number of polarizations as well as to the number of membranes. Allowing a special variant (c_λ) of rules of type (c) to delete symbols by sending them out, even only one membrane is enough.

Keywords: computational completeness, P systems, active membranes, polarizations

1 Introduction

Membrane systems are biologically motivated theoretical models of distributed and parallel computing, see [8] for a comprehensive overview and [11] for actual developments in the area. For P systems with active membranes and polarizations (charges $+, -, 0$ associated with the membranes, see [9]), the question of removing the polarizations without diminishing their computing power was formulated several times and was recently considered in various contexts (with the polarizations replaced by various other features, such as label changing – see, e.g., [2], [3]). Here we present another way for improving previous results: the number of polarizations can be decreased to two, without introducing new features. It is worth mentioning that the computational completeness is obtained for systems with the same types of rules as in [8], hence without using membrane division and membrane dissolution, even decreasing the number of membranes from three (Theorem 7.2.1 in [8]) to two. It remains as an open question whether polarizations can be completely removed.

In the following section we recall some basic notions from formal language theory and shortly prove a special normal form for graph-controlled grammars that we need in the proofs of the succeeding section; moreover, we introduce a slightly more general model of P systems with active membranes and arbitrary non-negative polarizations. In the third section we prove that P systems with active membranes and only two polarizations are already computationally complete - needing two membranes when using rules of types (a) and (c) as in the original definition and even only one membrane when using rules of types (a) and (c_λ) ; we finish with some open questions remaining for future research.

2 Prerequisites

The set of non-negative integers is denoted by $\mathbf{N}$. An *alphabet* V is a finite non-empty set of abstract *symbols*. Given V, the free monoid generated by V under the operation of concatenation is denoted by V^*; the *empty string* is denoted by λ, and $V^* - \{\lambda\}$ is denoted by V^+. By $\mid x \mid$ we denote the length of the word x over V. The family of recursively enumerable languages is denoted by RE; NRE denotes the family of recursively enumerable sets of non-negative integers.

Let $\{a_1, ..., a_n\}$ be an arbitrary alphabet; the number of occurrences of a symbol a_i in x is denoted by $\mid x \mid_{a_i}$; the *Parikh vector* associated with x with respect to $a_1, ..., a_n$ is $(\mid x \mid_{a_1}, ..., \mid x \mid_{a_n})$. The *Parikh image* of a language L over $\{a_1, ..., a_n\}$ is the set of all Parikh vectors of strings in L. For a family of languages FL, the family of Parikh images of languages in FL is denoted by $PsFL$. A (finite) multiset $\langle m_1, a_1\rangle ... \langle m_n, a_n\rangle$ with $m_i \in \mathbf{N}$, $1 \le i \le n$, is represented as any string x the Parikh vector of which with respect to $a_1, ..., a_n$ is $(m_1, ..., m_n)$.

In the following we will not distinguish between a vector $(m_1, ..., m_n)$, its representation by a multiset $\langle m_1, a_1\rangle ... \langle m_n, a_n\rangle$ or its representation by a string x with Parikh vector $(\mid x \mid_{a_1}, ..., \mid x \mid_{a_n}) = (m_1, ..., m_n)$.

For more notions as well as basic results from the theory of formal languages, the reader is referred to [4] and [10].

We now also recall the definition of a graph-controlled grammar and prove a special normal form needed later in the proofs given in this paper:

A *graph-controlled grammar* is a construct

$$G = (N, T, Lab, S, R, \{1\}, \{n\})$$

where N denotes the set of non-terminals, T is the set of terminal symbols, $Lab = \{1, ..., n\}$ is the set of labels, S is the start symbol, R is a finite set of rules that can be represented as a function from Lab to $P \times 2^{Lab} \times 2^{Lab}$, where P denotes the set of all context-free productions over the set N of non-terminal symbols and the set of terminal symbols T. A rule in R usually is written in the form $(i : p(i), \sigma(i), \varphi(i))$, where $\sigma(i)$ is called the success field and $\varphi(i)$ is called the failure field of the rule labelled by i; the context-free production $p(i)$ is of the form $A(i) \to w(i)$, where $A(i) \in N$ and $w(i) \in (N \cup T)^*$. Without loss of generality we not only assume that $N \cap Lab = \emptyset$ and that there is only one initial label (i.e., 1) and only one final label (i.e., n, with $\sigma(n) = \varphi(n) = \emptyset$), but we also may assume that if a computation has reached the final label n, then the obtained sentential form is terminal, i.e., it must not contain any non-terminal symbol.

As a special technical detail, without loss of generality we may assume any right-hand side $w(m)$ to contain at most one terminal symbol. Finally, again without loss of generality we may also assume that in the case of a string language, the terminal symbols are generated by G exactly in the correct sequence as they form a terminal word. All these features of a normal form for graph-controlled grammars, for example, follow from the constructions and results proved in [5], Theorem 6.

We now add one more feature to the normal form of graph-controlled grammars given above, i.e., from such a graph-controlled grammar G we now construct a graph-controlled grammar

$$G' = (N', T, Lab', S', R', \{0\}, \{n+1\})$$

with $N' = N \cup \{S', F\}$ and $Lab' = Lab \cup \{0, n+1, n+2\}$, which has the additional feature that all failure fields and all success fields in G' are non-empty:

First we construct the set of rules R'' from the set of rules R in the following way: For every rule $(i : p(i), \sigma(i), \varphi(i))$ in R we take the rule $\left(i : p(i), \sigma'(i), \varphi(i)'\right)$, where $\sigma'(i) = \sigma(i)$ for $\sigma(i) \neq \emptyset$ and $\sigma'(i) = \{n+2\}$ for $\sigma(i) = \emptyset$ as well as $\varphi'(i) = \varphi(i)$ for $\varphi(i) \neq \emptyset$ and $\varphi'(i) = \{n+2\}$ for $\varphi(i) = \emptyset$. Thus, we obtain

$$\begin{aligned} R' = R'' &\setminus \{(n : p(n), \emptyset, \emptyset)\} \\ &\cup \{(0 : S' \to SF, \{1\}, \{1\}), (n : F \to \lambda, \{n+1\}, \{n+1\})\} \\ &\cup \{(n+1 : F \to F, \{n+1\}, \{n+1\})\} \\ &\cup \{(n+2 : F \to F, \{n+2\}, \{n+2\})\}. \end{aligned}$$

We also assume the reader to be familiar with the basic elements of membrane computing, e.g., from [8] (details can be found at `http://psystems.disco.unimib.it`), in particular, with P systems with active membranes. For the sake of completeness, we recall the definition of P systems with active membranes for the case when only rules of types (a), (b), and (c) or (c_λ) are used; in a more general way as in the original definition, we allow the polarizations to be arbitrary non-negative integers:

A *P system system with active membranes* (of degree $m \geq 1$) is a construct of the form

$$\Pi = (O, E, \mu, w_1, \ldots, w_m, e_1, \ldots, e_m, R)$$

where O is the alphabet of objects, $E = \{0, ..., n-1\}$ with $n \geq 1$ is the set of electrical charges (polarizations), μ is the membrane structure (with m membranes, bijectively labelled with $1, 2, \ldots, m$; by H we denote the set of labels $\{1, 2, \ldots, m\}$), $w_1, \ldots, w_m$ are strings over O representing the multisets of objects occurring in the m regions of μ at the beginning of the computation, $e_1, \ldots, e_m$ are the polarizations at the beginning assigned to the membranes $1, \ldots, m$, and R is a finite set of rules of the following forms:

(a) $[a \to v]_h^i$, $a \in O$, $v \in O^*$, $h \in H$, $i \in E$
(evolution rule, used in parallel in the region of membrane h, provided that the polarization of the membrane is i);
(b) $a\,[\ ]_h^i \to [b]_h^j$, $a, b \in O$, $h \in H$, $i, j \in E$
(communication rule, sending an object into a membrane, possibly changing the polarization of the membrane);
(c) $[a]_h^i \to [\ ]_h^j\, b$, $a, b \in O$, $h \in H$, $i, j \in E$
(communication rule, sending an object out of a membrane, possibly changing the polarization of the membrane).
We shall also consider the following variant of rule type (c):
(c$_\lambda$) $[a]_h^i \to [\ ]_h^j\, b$, $a \in O$, $b \in O \cup \{\lambda\}$, $h \in H$, $i, j \in E$
(communication rule, sending an object out of a membrane or "killing" it by sending it through the membrane, possibly changing the polarization of the membrane).

Throughout this paper, we shall even use only communication rules $[a]_h^i \to [\ ]_h^j\, b$ with $a = b$ or $b = \lambda$.

The rules of types (b), (c), and (c_λ) are considered as involving the membrane, hence, we assume at most one such rule to be used for each membrane in a given step; the use of rules is maximally parallel, with the rules chosen in a non-deterministic manner. If no rule can be applied any more in the whole system, then we say that the computation halts. An output is associated with a halting computation – and only with halting computations – in the form of the objects sent into the environment during the computation; for the following definitions, we assume $\emptyset \subset D \subseteq \{a, b, c, c_\lambda\}$:

- If we consider only the number of symbols sent out during a halting computation, the set of all such numbers computed by a system Π is denoted by $N(\Pi)$. By $NOP_m\,(active_n, D)$ we denote the family of all sets $N(\Pi)$ computed by P systems with at most m membranes allowing for n polarizations, using rules of the types contained in D.

- If we distingish the different symbols sent out during a halting computation, the set of all such vectors of numbers computed by a system Π is denoted by $Ps(\Pi)$. By $PsOP_m\,(active_n, D)$ we denote the family of all sets $Ps(\Pi)$ computed by P systems with at most m membranes allowing for n polarizations, using rules of the types contained in D.
- If we consider the sequence of symbols sent out during a halting computation and interpret this sequence as a string, then the set of all such strings computed by a system Π is denoted by $L(\Pi)$. By $LOP_m\,(active_n, D)$ we denote the family of all languages $L(\Pi)$ computed by P systems with at most m membranes allowing for n polarizations, using rules of the types contained in D.

In this paper, we will use only two polarizations, 0 and 1, and this restriction will be indicated by writing $active_2$ in the notations defined above.

3 Computational Completeness with Two Polarizations

Stated in the notations of this paper, Theorem 1 from [7] says that $PsOP_3\,(active_3, \{a, b, c\}) = PsRE$. As announced above, we here not only improve this result with respect to the number of electrical charges (polarizations), but even with respect to the number of membranes, especially when allowing rules of type (c_λ) instead of rules (c) :

Theorem 1. $PsOP_1\,(active_2, \{a, c_\lambda\}) = PsRE.$

Proof. We only prove that for any recursively enumerable set of vectors of non-negative integers we can construct a P system with active membranes that generates a set of multisets representing L by using only one membrane, two polarizations, and rules of the types (a) and (c_λ).

We start with a graph-controlled grammar

$$G' = (N', T, Lab', S', R', \{0\}, \{n+1\})$$

that is in the normal form constructed in the preceding section and represents the given recursively enumerable set L of vectors.

We now construct a P system with active membranes of degree one

$$\Pi = (O, \{0,1\}, [_1\ ]_1, (S,0)\,(F,0)\,(1,0), 0, R_\Pi)$$

using only two polarizations 0 and 1 and rules of the form (a) and (c_λ) such that $PsP\,(\Pi) = L$:

$$\begin{aligned} O = {} & T \cup \{E\} \\ & \cup \{(B,l) \mid B \in N', 0 \le l \le 3n+1\} \\ & \cup \{(m,l) \mid 1 \le m \le n, 0 \le l < 3m\} \\ & \cup \{(\bar{m},l), (\hat{m},l) \mid 1 \le m \le n, 3m \le l \le 3n+1\} \\ & \cup \{m', m'', m''' \mid 1 \le m \le n\} \end{aligned}$$

The simulation of derivations in the graph-controlled grammar G' by derivations in the P system Π uses a colouring technique opening a "window" of length three for simulating the application of the rule $(m : p(m), \sigma(m), \varphi(m))$ currently to be applied. Basically, the labels $k \in Lab$ occur in the variants (k,l), $(\bar{k},l)$, or $\left(\hat{k},l\right)$ and the non-terminal symbols $B \in N'$ occur in the variants (B,l), $0 \leq l \leq 3n+1$.

In the following, we describe all the rules constituting R_{Π}; the label m runs from 1 to n :

- As long as membrane 1 (the skin) has polarization 0, the index l of the non-terminal symbols $B \in N'$ in (B,l) may be incremented:
  $[(B,l) \to (B,l+1)]_1^0$, $B \in N'$, $0 \leq l < 3n$.
- For each m, the index l of $m \in Lab$ in (m,l) is incremented until the index $3m-3$ is reached:
  $[(m,l) \to (m,l+1)]_1^0$, $0 \leq l < 3m-3$.
- Then we check whether $p(m)$ can be applied to the current contents of membrane 1; by polarizing the membrane we first prohibit the incrementation of the index l in the variables of the form (B,l) :
  $[(m,3m-3) \to (m,3m-2)\,E]_1^0$
  In the next step, all objects $(B,3m-2)$, $B \in N'$, in membrane 1 evolve to $(B,3m-1)$, whereas $(m,3m-2)$ evolves to $(m,3m-1)$ by applying the following rule:
  $[(m,3m-2) \to (m,3m-1)]_1^0$
  At the same time, E passes the skin membrane thereby changing its polarization from 0 to 1 :
  $[E]_1^0 \to [\ ]_1^1\,\lambda$
- With the polarization of the membrane being 1, the symbols now remain unchanged, yet – if possible – one copy of the object $A(m)$ (i.e., the non-terminal symbol on the left-hand side of the context-free production $p(m)$ in the rule labelled by m) has to pass the membrane resetting the polarization to 0:
  $[(A(m),3m-1)]_1^1 \to [\ ]_1^0\,\lambda$
  At the same time, the object $(m,3m-1)$ evolves according to the following rule:
  $[(m,3m-1) \to m']_1^1$
- In the next step m' evolves according to the polarization of the skin membrane (the polarization has stored the one-bit information whether $A(m)$ has been present or not):
  If the polarization is still 1, then m' evolves in two further steps to m''', where the symbol E generated in the first step then resets the polarization to 0 in the second step by passing the skin membrane:
  $[m' \to m''E]_1^1$,
  $[m'' \to m''']_1^1$,
  $[E]_1^1 \to [\ ]_1^0\,\lambda$

- As the polarization is 0 again, the symbols $(B, 3m-1)$, $B \in N'$, may evolve to $(B, 3m)$, whereas m' and m''', respectively, evolve to different symbols with index $3m$:
  $[m' \to (\bar{m}, 3m)]_1^0$
  $[m''' \to (\hat{m}, 3m)]_1^0$
- The symbols $(\bar{m}, 3m)$ and $(\hat{m}, 3m)$ until the end of a simulation cycle evolve in the same way as the basic objects (m, l) by incrementing the second parameter:
  $[(\bar{m}, l) \to (\bar{m}, l+1)]_1^0$, $3m \leq l < 3n$;
  $[(\hat{m}, l) \to (\hat{m}, l+1)]_1^0$, $3m \leq l < 3n$.
- At the end of a complete cycle, we finally extract the information stored in the symbols $\bar{m}$ and $\hat{m}$, respectively, and start a new cycle:
  $[(B, 3n) \to (B, 3n+1)]_1^0$ and
  $[(B, 3n+1) \to (B, 0)]_1^0$ for all $B \in N'$;
  $[(\hat{m}, 3n) \to (\hat{m}, 3n+1)]_1^0$ and
  $[(\bar{m}, 3n) \to (\bar{m}, 3n+1)\, h(w(m))]_1^0$, respectively, where
  $h : N' \cup T \to (N', 3n+1) \cup T$ is the morphism with
  $h(B) = (B, 3n+1)$ for all $B \in N'$ and $h(a) = a$ for all $a \in T$;
  observe that due to our assumptions about G', $w(m)$ (i.e., the right-hand side of the context-free production $p(m)$ in the rule labelled by m) contains at most one terminal symbol, hence, also $h(w(m))$ contains at most one terminal symbol a, which may leave the skin membrane by using the following rule:
  $[a]_1^0 \to [\ ]_1^0 a$
  The next cycle of simulating a derivation step in G' by Π starts after the application of one of the following rules:
  $[(\bar{m}, 3n+1) \to (k, 0)]_1^0$ for every $k \in \sigma(m) \setminus \{n+1, n+2\}$;
  $[(\hat{m}, 3n+1) \to (k, 0)]_1^0$ for every $k \in \varphi(m) \setminus \{n+1, n+2\}$.
  In case that the label of the "trap" $n+2$ is reached then we can immediately enter an infinite loop due to the additional symbol F still present in its indexed variants (F, l), $0 \leq l \leq 3n+1$:
  $[(\bar{m}, 3n+1) \to \lambda]_1^0$ for every m with $n+2 \in \sigma(m)$;
  $[(\hat{m}, 3n+1) \to \lambda]_1^0$ for every m with $n+2 \in \varphi(m)$.
- The simulation of a derivation in G' by Π may successfully end if we can apply
  $[(\bar{n}, 3n+1) \to \lambda]_1^0$.
  Due to our assumptions for G', after applying this rule in Π no non-terminal symbol can appear any more (observe that by simulating the rule labelled by n the additional symbol F has disappeared, too); hence, in case of termination we finish with an empty membrane.

The construction given above completely describes the set of rules R_Π of the P system with active membranes Π.

From the explanations given above, it is obvious that the P system with active membranes Π defined above exactly generates a set of multisets that represents the same set of vectors as the given graph-controlled grammar. This observation completes the proof. □

We could also consider P systems with extensions, i.e., in the constructions above we could read every λ there representing the empty word as a special non-terminal symbol not being taken into account when considering the resulting multisets; in such a case, even using only rules of the form (c) instead of rules of the form (c_λ) would already yield computational completeness in only one membrane. Yet we do not follow this direction any further, as the related results are obvious and directly follow from the proofs given in this section. Instead, we prove that even with the original types of rules (a) and (c) we only need one additional membrane being only used for filtering out the non-terminal symbols having passed the inner membrane:

Theorem 2. $PsOP_2\left(active_2, \{a, c\}\right) = PsRE$.

Proof. Again we only prove $PsRE \subseteq PsOP_2\left(active_2, \{a, c\}\right)$ starting with a graph-controlled grammar

$$G' = \left(N', T, Lab', S', R', \{0\}, \{n+1\}\right)$$

in the normal form as constructed in the preceding section which generates (a string language representing) the given recursively enumerable set L of vectors.

We now construct a P system with active membranes of degree two

$$\Pi' = \left(O, \{0,1\}, [_1[_2\ \ ]_2]_1, \lambda, (S,0)\,(F,0)\,(1,0), 0, 0, R'_\Pi\right)$$

using only two polarizations 0 and 1 and rules of the form (a) and (c) which generates (a set of multisets representing) L. The set of objects O is identical with that one in the preceding proof.

The simulation of derivations in the graph-controlled grammar G' by derivations in the P system Π' again uses the same colouring technique as described in the previous proof; the simulation of a derivation step is carried out in membrane 2 by opening a "window" of length three for simulating the application of the rule $(m : p\,(m)\,, \sigma\,(m)\,, \varphi\,(m))$ currently to be applied. The non-terminal symbols sent out through membrane 2 remain unchanged in this region enclosed by the skin membrane. On the other hand, the terminal symbols having passed membrane 2, in the succeeding step leave the system by immediately passing through the skin membrane. Hence, from R_Π constructed in the preceding proof we immediately obtain R'_Π by the following procedure:

1. in all rules of R_Π, replace label 1 by label 2, thus obtaining R''_Π;
2. replace each rule $[\alpha]_2^1 \to [\ ]_2^0\,\lambda$, for some $\alpha \in O$, from R''_Π by $[\alpha]_2^1 \to [\ ]_2^0\,\alpha$, thus obtaining R'''_Π;
3. $R'_\Pi = R'''_\Pi \cup \left\{[a]_1^0 \to [\ ]_1^0\,a \mid a \in T\right\}$.

The rules of the form $[a]_1^0 \rightarrow [\ ]_1^0 a$ are the only rules affecting the skin membrane (without changing its polarity); moreover, there are no evolution rules in region 1, i.e., the other (non-terminal) symbols coming through membrane 2 are never changed in region 1, they remain there as a kind of "garbage".

From the construction given above, it is obvious that the P system with active membranes Π' in a similar way as the P system with active membranes Π constructed in the proof of the preceding theorem exactly generates a set of multisets that represents the same set of vectors as the given graph-controlled grammar, which observation completes the proof. □

The following two corollaries are immediate consequences of the two preceding theorems, i.e., obviously also when considering only the number of symbols sent out through the skin membrane without distinguishing between different symbols we obtain the corresponding results:

Corollary 1. $NOP_1\,(active_2, \{a, c_\lambda\}) = NRE.$

Even when taking the original definitions from [8], we can considerably improve the result stated in Theorem 7.2.1 there, which in the notations defined in this paper says $NOP_3\,(active_3, \{a, b, c\}) = NRE$, i.e., we can improve the result with respect to the number of polarizations as well as to the number of membranes:

Corollary 2. $NOP_2\,(active_2, \{a, c\}) = NRE.$

So far, the terminal symbols – as the result of a successful computation – have been sent out of the skin membrane without regarding the order of their appearance; regarding the sequence of symbols sent out during a successful (i.e., halting) computation as a string we obtain languages:

Corollary 3. $LOP_1\,(active_2, \{a, c_\lambda\}) = LOP_2\,(active_2, \{a, c\}) = RE.$

Proof. We only prove that any recursively enumerable language can be generated by a P system with active membranes using only one (two) membrane(s), two polarizations, and rules of the form (a) and (c_λ) $((c))$. The proof immediately follows from Theorem 1 (Theorem 2) - due to the special feature (based on the details of the proof of Theorem 6 in [5]) of the given graph-controlled graph grammar in normal form being constructed in such a way that the symbols of a terminal string are generated symbol by symbol in the same order as they form this string. Hence, in the simulating P system the terminal symbols pass the skin membrane (pass the second membrane of the simulating P system and one step later are sent out through the skin membrane) in just the same sequence as they are generated by the graph-controlled grammar. This observation already completes the proof. □

In addition to the generative P systems with active membranes, we could also consider the following variants:

- *Computing* P systems with active membranes start with multisets of objects given in a specified input membrane and then (by halting computations) compute functions. Again similar results as those stated in Theorems 1 and 2 as well as Corollaries 1 and 2 hold true.
- *Accepting* P systems with active membranes accept multisets of objects given in a specified input membrane by halting computations; now one membrane is enough even in the case of rules of type (a) and (c), because we do not consider any ouput, but simply accept by halting (even with an empty membrane as follows from the construction in the proof of Theorem 1).

Without going into the very details of the proofs we just mention that for computing P systems with active membranes as well as for accepting P systems with active membranes we simulate *deterministic* graph-controlled grammars (each success field and each failure field contains exactly one element). The input values are given as multisets over an input alphabet which is considered to be a subset of the non-terminal alphabet of the deterministic graph-controlled grammar. According to the constructions given in the proofs of Theorems 1 and 2, the simulation of the deterministic graph-controlled grammar by the corresponding (computing, accepting) P system with active membranes is deterministic, too, which is a very important feature in the area of P systems (e.g., see [6]).

At the end of this section, we list some problems left open despite the partly already optimal results proved above:

- What happens if we allow only rules of the types (a) and (c) in one membrane, but possibly an unbounded number of polarizations, i.e., how can we characterize $PsOP_1\left(active_n, \{a, c\}\right)$ for $n \geq 1$?
- How can we characterize $PsOP_m\left(active_1, \{a, c_\lambda\}\right)$, $m \geq 1$?
- Can we at least characterize $PsOP_1\left(active_1, \{a, \gamma\}\right)$ for $\gamma \in \{c, c_\lambda\}$?

Acknowledgements

This paper was initiated during the Brainstorming Week on Membrane Computing taking place in Sevilla during the first week of February, 2004. Even some more detailed proofs as well as an algorithm for deterministically deciding `SAT` in linear time using only two polarizations and global rules of types (a) and (c) as well as membrane division can be found in [1].

The authors acknowledge IST-2001-32008 project "MolCoNet". The first author is supported by the project TIC2002-04220-C03-02 of the Research Group of Mathematical Linguistics, Tarragona; he also acknowledges the Moldovan Research and Development Association (MRDA) and the U.S. Civilian Research and Development Foundation (CRDF), Award No. MM2-3034.

References

1. A. Alhazov, R. Freund, Gh. Păun: P Systems with active membranes and two polarizations. In: Gh. Păun, A. Riscos Nuñez, A. Romero Jiménez, F. Sancho Caparrini (Eds.): *Second Week on Membrane Computing*. Sevilla, Spain, Feb. 2-7, 2004. Dept. of Computer Sciences and Artificial Intelligence, Univ. of Sevilla Tech. Report 01/2004 (2004) 20–36
2. A. Alhazov, L. Pan: Polarizationless P systems with active membranes. To appear in *Grammars* (2004)
3. A. Alhazov, L. Pan, Gh. Păun: Trading polarizations for labels in P systems with active membranes. Submitted (2003)
4. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, Berlin (1989)
5. R. Freund, C. Martín-Vide, Gh. Păun: From regulated rewriting to computing with membranes: collapsing hierarchies. *Theoretical Computer Science* **312** (2004) 143–188
6. R. Freund, Gh. Păun: Deterministic P systems. Submitted (2004)
7. M. Madhu, K. Krithivasan: Improved results about the universality of P systems. *Bulletin of the EATCS* **76** (2002) 162–168
8. Gh. Păun: *Computing with Membranes: An Introduction*. Springer, Berlin (2002)
9. Gh. Paun: Computing with membranes - a variant: P systems with polarized membranes. *Intern. J. of Foundations of Computer Science* **11**, 1 (2000) 167–182 and CDMTCS TR **098**, Univ. of Auckland (1999)
10. A. Salomaa, G. Rozenberg (Eds.): *Handbook of Formal Languages*. Springer-Verlag, Berlin (1997)
11. The P Systems Web Page: `http://psystems.disco.unimib.it`

Computing with a Distributed Reaction-Diffusion Model

S. Bandini[1], G. Mauri[1], G. Pavesi[2], and C. Simone[1]

[1] Dept. of Computer Science, Systems and Communication
University of Milan–Bicocca
Via Bicocca degli Arcimboldi 8
Milan, Italy
{bandini,mauri,simone}@disco.unimib.it

[2] Dept. of Computer Science and Communication (D.I.Co)
University of Milan
Via Comelico 39
Milan, Italy
pavesi@dico.unimi.it

Abstract. Reaction–diffusion models are commonly used to describe dynamical processes in complex physical, chemical and biological systems. Applications of these models range from pattern formation or epidemic spreads to natural selection through ecological systems and percolation systems. Reaction refers to phenomena where two or more entities become in contact and modify their state as a consequence of this fact. Diffusion implies the existence of a space where the involved entities are situated and can move. The Reaction–Diffusion Machine is a computational model we previously introduced inspired by reaction diffusion phenomena. In this work, we prove that a Deterministic Turing Machine can be simulated by a Reaction-Diffusion Machine.

1 Introduction

Reaction–diffusion models are commonly used to describe dynamical processes in complex physical, chemical and biological systems. Applications of these models range from pattern formation [1] or epidemic spreads to natural selection through ecological systems [2] and percolation systems [3]. Reaction refers to phenomena where two or more entities (agents) become in contact and modify their state in consequence of this fact. Diffusion implies the existence of a space where the involved agents are situated and can move.

A computational model inspired to reaction diffusion phenomena, called Reaction-Diffusion Machine (RDM), has been first introduced in [4] and further developed in [5]. It allows for the simulation of complex systems in which entities react locally with each other and with the environment, and the global system behaviour emerges from the local behaviour of the composing entities. In RDM the control is fully distributed. Agent behaviour is determined by a local "computation" based on their position and sensitivity to fields as well as on reaction and diffusion patterns characterising their type. This paper is a first step towards the theoretical assessment of the RDM, through the comparison with standard computational models such as deterministic Turing machines.

2 Turing Machines

Let us first recall the well–known definition of *Turing machine* (see for example [6]). A *deterministic one tape Turing machine* (DTM) consists of a *finite state control*, a *read–write head*, and a *tape*, made of a bi–infinite sequence of *tape cells* numbered $\ldots, -2, -1, 0, 1, 2, \ldots$. In addition, a DTM is characterized by: a finite tape alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$, which is taken as an ordered set, and a blank symbol b not belonging to Σ; a finite set of states Q, including a distinguished start state q_0 and two distinguished final states q_Y and q_N; and a state transition function $\delta : Q - \{q_Y, q_N\} \times \Sigma \rightarrow Q \times \Sigma \times \Delta$, where $\Delta = \{-1, 1\}$.

The *input* to the DTM is a string $x \in \Sigma^*$. The string x is written in tape cells 1 through $|x|$, one symbol per cell. All other cells initially contain the blank symbol. The DTM starts in state q_0 with the tape head scanning cell 1. The computation proceeds in a step–by–step fashion. If the current state is either q_Y or q_N the computation stops, and the DTM accepts x if $q = q_Y$, or rejects it if $q = q_N$. Otherwise, $q \in Q - \{q_Y, q_N\}$, and some symbol s is in the cell being scanned. Now, suppose that $\delta(q, s) = (q', s', \Delta)$. The tape head erases s, writes s' in its place, and moves either one cell to the left if $\Delta = -1$, or to the right if $\Delta = 1$. At the same time, the control changes its state from q to q'. This completes one step of the DTM.

3 Reaction-Diffusion Machines

A *Reaction–Diffusion Machine* (RDM) is a tuple $\langle S, \mathbf{F}, T, C_0, \mathcal{R} \rangle$ [5], where:

1. S is a *space*, represented as a connected undirected graph $G = (V, E)$;
2. $T = \{\tau_1, \ldots, \tau_n\}$ is a set of entity *types*; a finite set of *states* X_{τ_i} is associated with each type $\tau_i \in T$;
3. $\mathbf{F}$ is a vector of *fields* active in the space; fields are defined by:
   (a) a set of values W_i associated with each field $F_i \in \mathbf{F}$;
   (b) a set of *field source functions* $\Phi = \{\phi_{\tau_1}, \ldots, \phi_{\tau_n}\}$ associated with each type $\tau \in T$; $\forall i$, function $\phi_{\tau_i} : X_{\tau_i} \rightarrow W_1 \cup \bot \times \ldots \times W_k \cup \bot$ defines when an entity of type τ_i can emit each of the fields, and the intensity of the fields ($\bot$ if the field is not emitted), according to its state; the cumulative effect of fields of the same type emitted by different sources reaching the same node can be modeled by defining suitable *field composition* functions;
4. C_0 is the initial configuration;
5. $\mathcal{R}$ is a set of *rules*, defining how and when entities interact.

The space of the RDM is populated by *entities*. An entity e is denoted by a triple $\langle p, \tau_e, x_e \rangle$, where $p \in V$ is the node where the entity is located, $\tau_e \in T$ is the entity *type*, and $x_e \in X_{\tau_e}$ is its state, belonging to the finite set of possible states associated with its type. At any given time point, each node of the graph contains at most one entity (or can be vacant). Entities can emit fields (according to field source functions ϕ_i) that propagate throughout the space, can perceive fields, according to their type and state, can react with neighboring entities (located in adjacent nodes of the graph), and can move in the space (again, as a consequence of the perception of fields).

The initial configuration C_0 defines the entities present in the space, their position, their type, and their state. Starting from the initial configuration, the evolution of the RDM is governed by the set of rules $\mathcal{R}$. There are four types of rules, determining:

1. if, and how, neighboring entities interact with each other when located in adjacent positions in the graph (*reaction* rules denoted by r^R);
2. how fields propagate in the space (*field diffusion* rules, r^F); that is, the intensity of a field can change according to the distance in the space from the position of the emitting entity, and the initial emission value;
3. how entities change their state (*trigger* rules, r^T) because of the perception of fields;
4. how entities move in the space (*transport* rules, r^{TR}) because of the perception of fields.

Thus, rules determine the global evolution of the RDM, that is, how the single entities change their state because of local interactions and/or the perception of fields (i.e. the effect of long–distance interactions), how entities move in the space, and which fields are active in the space and their respective intensity.

The computation of the RDM proceeds in a series of discrete time steps, where entities change their state simultaneously. At each step:

1. Each entity determines whether a reaction or a trigger rule can be applied, according to the presence of neighboring entities and/or the perception of fields. If a rule can be applied, the entity changes its state accordingly. In case more than a single rule can be applied, different approaches can be followed: for example, we could define priority values associated with each rule, and the entity applies the one with highest priority, or the entity could simply choose a rule at random, and so on. If no rule can be applied, the entity keeps its state unchanged.
2. Once the state of the entities has been changed, each entity determines whether a movement rule can be applied, and changes its position accordingly. Conflict–resolving criteria for different rules and for competition (that is, when two entities try to move to the same graph node) can be defined also in this case.
3. Each entity emits one or more fields, according to its state and field source functions.
4. Finally, fields emitted by the entities are propagated throughout the space, according to field diffusion rules.

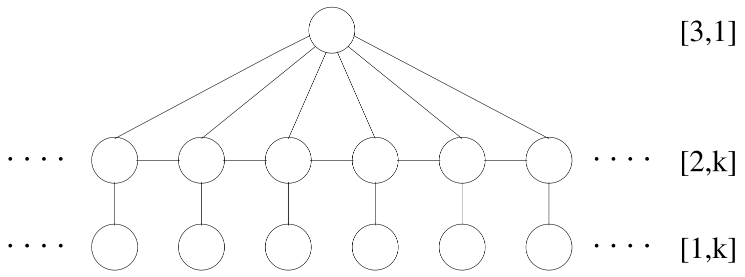


Fig. 1. The TRDM space.

Once all the sub–steps just described have been performed, the RDM has entered a new state, defined by the position and state of the entities as well as the fields' intensity value in each of the nodes of the graph.

In the following, we explicitly define a RDM designed for the simulation of a deterministing Turing machine. We refer the reader to [5] for a more detailed definition of the RDM and its properties.

4 The TRDM

In order to define the RDM simulating the DTM, that we will call TRDM, each of the RDM components has to be suitably designed. As we will see, the rules have been defined explicitly in order to avoid any possible conflict among rules and in entity movement, that is, to obtain a deterministic behaviour.

The nodes of the graph representing the TRDM space can be split into three different levels, as shown in Fig. 1. We will denote by $[1, k]$, $k \in (-\infty, +\infty)$ the k–th node of the first level, that simulates the tape of the Turing machine; by $[2, k]$ the k–th node of the second level, that reproduces the possible positions of the tape head; and by $[3, 1]$ the unique node on the third level, that contains the finite state control. Node $[1, j]$ is connected only to $[2, j]$, while all nodes $[2, j]$ are also connected to $[2, j - 1]$, $[2, j + 1]$ and also to node $[3, 1]$.

Entities represent the different elements composing the Turing machine, i.e. tape, head and finite state control, and can thus belong to three different types; hence, $T = \{\tau_1, \tau_2, \tau_3\}$. Each level of the space is therefore populated by entities of the same type. Let $Sym = \{-1, 0, 1, \dots, |\Sigma|\}$ be a set of integer numbers corresponding to the symbols of Σ plus the blank symbol (-1) and a "no symbol" identifier (0). The set of states associated with each type are:

1. $X_{\tau_1} = \{Sym - \{0\}\} \times \mathbf{Z}$, where $\mathbf{Z}$ denotes the set of integer numbers. That is, the states of entities of type τ_1 correspond to the alphabet symbols, or to the blank symbol b. The second component of the state is used by the entities to know their position in the bottom level of the graph (i.e. the cell number). We will call entities of this type *tape entities*.

2. $X_{\tau_2} = \mathit{Sym} \times \mathit{Op} \times \Delta \cup \{0\} \times \mathbf{Z}$, with $\mathit{Op} = \{R, W, M\}$. The unique entity of type τ_2 simulating the tape head of the DTM assumes states representing the symbol to be read or written (Sym), the operation to be performed (Op, that can be read, write, or move), and how the head moves (Δ). The integer number of the last state component is used by the entity to keep track of the last position along the tape before the current one. We will call this entity *head entity.*
3. $X_{\tau_3} = Q \times \mathit{Sym} \times \Delta \cup \{0\} \times \{R, W\}$. The unique entity of type τ_3 represents the finite control of the DTM; Q is the state set of the finite control; Sym and Δ are used to notify to the tape head the symbol that has to be written and the subsequent movement; R and W indicate which operation the control is waiting for to be completed. We will call this entity *control entity.*

Three fields, F_1, F_2, and F_3 are active in the TRDM space. They are used by the tape head entity to communicate with the finite state control entity and the tape symbol entities. The sets of possible values for the three fields are $W_1 = \mathbf{Z}$, $W_2 = \mathit{Sym}$, $W_3 = \Delta \times W_2$. The field source functions, defining when and how the different entity types emit the fields are:

1. $\forall \langle j, k \rangle \in X_{\tau_1}$, $\phi_{\tau_1}(\langle j, k \rangle) = \langle k, \perp, \perp \rangle$; entities of type τ_1 (representing tape symbols) emit only field F_1 (the values for F_2 and F_3 are undefined), with intensity corresponding to their position in the first level of the graph (i.e. their cell).
2. $\phi_{\tau_2}(\langle i, Y, d, k \rangle) = \langle \perp, i, \perp \rangle$, if $i > 0$ and $Y = R$, $\phi_{\tau_2}(\langle i, Y, d, k \rangle) = \langle \perp, 0, \perp \rangle$ if $Y = M$; $\phi_{\tau_2}(\langle i, Y, d, k \rangle) = \langle \perp, \perp, \perp \rangle$ if $Y = W$. The head entity of type τ_2 emits only field F_2 to propagate the symbols read from the tape to the finite state control (i), and to signal to the latter that a write operation has been completed (in this case, $F_2 = 0$).
3. $\phi_{\tau_3}(\langle q, d, j, W \rangle) = \langle \perp, \perp, \langle d, j \rangle \rangle$ if $q \neq \{q_Y, q_N, q_0\}$, and $d, j \neq 0$, $\langle \perp, \perp, \perp \rangle$ otherwise. When it is not in one of the halting states q_Y and q_N or in the initial state, the control entity emits field F_3 to signal to the head entity which symbol has to be written on the tape (j) and where it has to move (d).

In the TRDM, we assume that fields generated by entities propagate only to nodes within distance one from the source node, keeping the same intensity value, and can be perceived by all the entities, without explicitly specifying field diffusion rules. Moreover, no field composition functions are needed. The other rules of the TRDM have been designed to simulate the computation of the DTM. By $F_i[p]$ we denote the value of field F_i in node $p \in V$. In the following, we will define the rules by describing the conditions that have to be satisfied by one or more entities (according to entities type, position, and state), and possible field values in some space positions, and by describing the change in the state or position of the entities involved.

Reaction Rules

$$r_1^R = \frac{\langle [1,k], \tau_1, \langle j,k \rangle \rangle, \langle [2,k], \tau_2, \langle 0, R, 0, k' \rangle \rangle}{\langle [1,k], \tau_1, \langle j,k \rangle \rangle, \langle [2,k], \tau_2, \langle j, R, 0, k' \rangle \rangle} \tag{1}$$

Rule r_1^R is applied to the head entity (type τ_2) and a tape entity (τ_1) when located in the adjacent nodes of the graph (at the tape and head levels, in position k). It simulates a read operation of the tape head of the DTM, and changes the head entity state according to the symbol read (j).

$$r_2^R = \frac{\langle [1,k], \tau_1, \langle i,k \rangle \rangle, \langle [2,k], \tau_2, \langle j, W, d, k' \rangle \rangle}{\langle [1,k], \tau_1, \langle j,k \rangle \rangle, \langle [2,k], \tau_2, \langle 0, M, d, k \rangle \rangle} \tag{2}$$

Rule r_2^R involves the head entity and a tape entity in an adjacent position, and simulates a write operation of the head of the DTM. In fact, the tape entity involved in the reaction changes its state from $\langle i, k \rangle$ to $\langle j, k \rangle$, while the head entity is ready to move to a new position after writing on the tape.

Trigger Rules

$$r_1^T = \frac{\langle [3,1], \tau_3, \langle q, 0, 0, R \rangle \rangle, F_2[3,1] = i}{\langle [3,1], \tau_3, \langle \delta(q, \sigma_i), W \rangle \rangle} \tag{3}$$

Rule r_1^T simulates the state transition of the DTM state control, by changing the state of the entity located at node $[3,1]$. Field F_2 emitted by the head entity propagates the "code" of the symbol just read (i) from the tape. The control entity perceives it and changes its state according to the transition function δ from q to $\delta(q, \sigma_i)$. Notice that in the new state the control entity is no longer sensitive to field F_2, since it changes the "next operation" value from R to W.

$$r_2^T = \frac{\langle [2,k], \tau_2, \langle i, R, 0, k' \rangle \rangle, F_3[2,k] = \langle d, j \rangle}{\langle [2,k], \tau_2, \langle j, W, d, k' \rangle \rangle} \tag{4}$$

Rule r_2^T is used to propagate from the finite state control to the tape head entity the symbol that has to be written on the tape (j), and how the head has to move (d). The tape head entity changes its state and gets ready to write symbol σ_j.

$$r_3^T = \frac{\langle [3,1], \tau_3, \langle q, j, d, W \rangle \rangle, F_2[3,1] = 0}{\langle [3,1], \tau_3, \langle q, 0, 0, R \rangle \rangle} \tag{5}$$

This rule is used to simulate an "acknowledgement" from the tape head to the state control. The head entity has just written the symbol emitted by the control entity with rule r_2^R, and is emitting field F_2 with intensity 0. The control entity perceives the field, and changes its state in order to wait for the next read operation to be completed, no longer emitting field F_3.

$$r_4^T = \frac{\langle [2,k], \tau_2, \langle 0, M, d, k' \rangle \rangle, F_1[2,k] = k, k \neq k'}{\langle [2,k], \tau_2, \langle 0, R, 0, k \rangle \rangle} \tag{6}$$

Rule r_4^T is applied to the tape head entity after it has moved to a new position. The entity realizes that it has just arrived in a new position (by checking the value of the field emitted by the tape symbol entity), and restores itself in a "read" state in order to perform a read operation. Note that, after the application of this rule, the head entity is no longer sensitive to field F_1.

Transport Rules

$$r_1^{TR} = \frac{\langle [2,k], \tau_2, \langle 0, M, d, k' \rangle\rangle, F_1[2,k] = k, k = k'}{\langle [2, k+d], \tau_2, \langle 0, M, d, k' \rangle\rangle} \tag{7}$$

This rule makes the tape head move. The entity is ready to move (it is in the "move" state denoted by M), perceives the position along the tape emitted by the tape entity with field F_1, and moves according to the d value that has been previously communicated by the control entity.

5 The Simulation

A *configuration* of the DTM is a triple $C^T = \langle q, c, T\rangle$, where q is the state of the finite control, c is the cell where the tape head is located, and $T = \{\dots, t_{-1}, t_0, t_1, \dots\}$ is the sequence of symbols on the tape, either belonging to the alphabet Σ or blank. An *halting* configuration for the DTM is a configuration with the control in one of the halting states. An *accepting* configuration is an halting configuration with the control in state q_Y; an halting configuration with the control in state q_N is a rejecting configuration. A computation of the DTM can be expressed as a sequence of configurations $C_0^T, C_1^T, \dots, C_n^T$. Analogously, a configuration of the TRDM, denoted by C^R is described by the state and position of the entities active in the space, and by the set of fields that are active. The function $Loc : V \rightarrow T \times E \cup \{\perp\}$ associates with each node of the graph the type and the state of the entity located in the node ($\perp$, if the node is empty). Entities emit fields according to the field source functions. An *halting* configuration of the TRDM is a configuration where no rule can be applied. *Accepting* and *rejecting* configurations are halting configurations with the control entity in state $\langle q_Y, \cdot, \cdot, \cdot\rangle$ and $\langle q_N, \cdot, \cdot, \cdot\rangle$, respectively.

Let us suppose that the DTM has been given the input string $s = s_1 s_2 \dots s_n$. The string is positioned on the tape starting from cell 1. All other tape cells contain the blank symbol. The initial configuration of the DTM is $C_0^T = \langle q_0, 1, T\rangle$, with the control in the initial state q_0, the tape head scanning cell 1 of the tape, and $T = \{\dots, b, s_1, s_2, \dots, s_n, b, \dots\}$. The initial configuration of the TRDM reproduces the configuration of the DTM:

1. if $1 \le k \le n$, $Loc[1,k] = \langle \tau_1, \langle j, k\rangle\rangle$, where j is such that $\sigma_j = s_k$; otherwise, $Loc[1,k] = \langle \tau_1, \langle -1, k\rangle\rangle$;
2. $Loc[2,1] = \langle \tau_2, \langle 0, R, 0, 0\rangle\rangle$; $Loc[2,k] = \perp$, $\forall k \neq 1$.
3. $Loc[3,1] = \langle \tau_3, \langle q_0, 0, 0, R\rangle\rangle$;

In the same way, it is possible to define, for every configuration $C^T = \{q, c, T\}$ of the DTM, the corresponding configuration of the TRDM, where:

1. $Loc[3,1] = \langle \tau_3, \langle q, 0, 0, R\rangle\rangle$.
2. $Loc[2,c] = \langle \tau_2, \langle 0, R, 0, c\rangle\rangle$; $Loc[2,k] = \bot$, $\forall k \neq c$;
3. $\forall i$ such that $t_i = \sigma_j$, $Loc[1,i] = \langle \tau_1, \langle j, i\rangle\rangle$; $\forall i$ such that $t_i = b$, $Loc[1,i] = \langle \tau_1, \langle -1, i\rangle\rangle$.

Given a configuration C^T of the DTM, we denote by $Conf(C^T)$ the corresponding configuration of the TRDM.

Potentially, starting from the initial configuration, different sequences of rule applications could lead to different sequences of configurations, i.e. different behaviours of the TRDM. However, the rules have been defined in order to obtain a deterministic behaviour, as we are going to show. According to the standard terminology, a configuration C_k^R for the TRDM is *reachable* from the initial configuration C_0^R if there is a sequence of applications of rules leading from C_0^R to C_k^R. In the same way, a configuration C_k^T of the DTM is reachable from the initial configuration C_0^T if there exists a sequence of state transitions leading to C_k^T.

Lemma 1. *In every reachable configuration of the TRDM:*

1. *only one rule can be applied, and once a rule has been applied, it will be applied again only after the application of all the other rules;*
2. *if the tape head entity is in state $\langle 0, R, 0, k\rangle$, the configuration of the TRDM corresponds to a reachable non halting configuration of the DTM.*

Proof (sketch) In the initial configuration of the TRDM, no field is active, and trigger and transport rules cannot be applied. Since the head entity is in state $\langle 0, R, 0, 0\rangle$, it is not sensitive to field F_1 that the tape entities start to emit.

Therefore, the only rule that can be applied is r_1^R. At this point, the head entity enters state $\langle i, R, 0, 0\rangle$, such that $s_1 = \sigma_i$, and starts to emit field F_2 with intensity i. While rule r_1^R can no longer be applied, rule r_1^T can now be activated, determining the effect of field F_2 on the control entity. The control entity enters state $\langle\langle \delta(q_0, \sigma_i)\rangle, W\rangle$, can no longer perceive F_2, and starts to emit field F_3.

The only rule that can be applied now is the trigger rule r_2^T corresponding to F_3, that changes the state of the head entity from $\langle i, R, 0, 0\rangle$ to $\langle j, W, d, 0\rangle$ (j corresponds to symbol σ_j that has to be written on tape).

At this point, F_3 is no longer perceived by the head entity, that stops emitting field F_2 for effect of the change of state: the only rule that can be applied is r_2^R between the head entity and the tape entity in $[1,1]$, that change their state, respectively, to $\langle 0, M, d, 1\rangle$ and $\langle j, 1\rangle$. Now, the head entity emits again field F_2, that activates rule r_3^T for the control entity, that stops emitting F_3 and enters state $\langle q', 0, 0, R\rangle$.

Rule r_1^{TR} is the only rule that can now be applied, and the head entity moves from node $[2,1]$ to node $[2, 1+d]$. Once arrived in the new position, it perceives the field emitted by the tape entity and changes its state to $\langle 0, R, 0, 1\rangle$ for effect of rule r_4^T.

In other words, the order of application of the rules reproduces the single DTM operations, as shown in Fig 2. Rule r_1^R reproduces the tape head reading a symbol; rule r_1^T propagates the symbol read to the control entity, that changes its state accordingly reproducing the DTM transition function; rules r_2^T, r_2^R and r_3^T are used to reproduce the writing of a symbol on the tape; finally rules r_1^{TR} and r_4^T simulate the movement of the tape head.

It is straightforward to see that the configuration reached by the TRDM at this point corresponds to a reachable configuration of the DTM, that is, the one reached after one computation step (C_1^T), and again the only rule that can be applied is r_1^R, corresponding to reading a symbol from the tape, as at the beginning of the simulation. The same argument holds for any reachable configuration of the DTM. In particular, given any given configuration $C_k^R = \mathit{Conf}(C_k^T)$, the TRDM moves to the configuration $C_{k+1}^R = \mathit{Conf}(C_{k+1}^T)$, again by following a unique series of rule application identical to the one leading from $\mathit{Conf}(C_0^T)$ to $\mathit{Conf}(C_1^T)$. □

According to the previous lemma, when the TRDM is initialized in configuration $C_0^R = \mathit{Conf}(C_0^T)$ corresponding to the initial configuration of the DTM, it evolves deterministically through a sequence of configurations $C_1^R \dots C_k^R$ such that $C_1^R = \mathit{Conf}(C_1^T) \dots C_k^R = \mathit{Conf}(C_k^T)$, corresponding to the computation of the DTM.

To complete the simulation we have to prove the following:

Theorem 1. *Given an initial configuration for the DTM and the corresponding initial configuration for the TRDM, the TRDM reaches an halting configuration iff the DTM halts. Moreover, the TRDM reaches an accepting configuration iff the DTM accepts the input string.*

Proof. Let $C_0^T, C_1^T, \dots, C_{(h-1)}^T$ the sequence of configurations resulting from the computation of the DTM leading to the halting configuration C_h^T. We suppose without loss of generality that C_h^T is an accepting configuration, that is, with the control in state q_Y. According to Lemma 1, when the DTM is in configuration $C_{(h-1)}^T = \langle q_{(h-1)}, k, T\rangle$, the TRDM is in the corresponding configuration $C_{(h-1)}^R = \mathit{Conf}(C_{(h-1)}^T)$, where the tape head entity is located on node $[2, k]$ and is in state $\langle 0, R, 0, k\rangle$, the tape symbol entities are in states corresponding to the symbols contained in the DTM tape at step $(h-1)$, and the control entity is in state $\langle q_{(h-1)}, 0, 0, R\rangle$. At this point, the sequence of application of the rules is the one sketched above for the transition from C_0^R to C_1^R, until the application of rule r_2^T. The difference is that now the control entity has entered the state $\langle \delta(q_{(h-1)}, s_k), W\rangle$. Since C_h^T is an accepting configuration for the DTM, we have that $\delta(q_{(h-1)}, s_k) = q_Y$. Therefore, the control entity enters the state $\langle q_Y, 0, 0, W\rangle$. According to its field source function, it cannot emit any field when it enters a state corresponding to q_Y. Without field F_3, no other rule can be applied at this point, and the TRDM halts in an accepting configuration. □

$$\begin{tikzcd}
C_k^T \arrow[dddddddd, "\delta"] & C_k^R = \mathit{Conf}(C_k^T) \arrow[d] & \\
 & r_1^R \arrow[d] & \text{Reading of the tape symbol} \\
 & r_1^T \arrow[d] & \text{State transition of the finite control} \\
 & r_2^T \arrow[d] & \text{Instructions for the tape head} \\
 & r_2^R \arrow[d] & \text{Writing a symbol on the tape} \\
 & r_3^T \arrow[d] & \text{Write operation completed} \\
 & r_1^{TR} \arrow[d] & \text{Movement of the tape head} \\
 & r_4^T \arrow[d] & \text{DTM transition completed} \\
C_{k+1}^T & C_{k+1}^R = \mathit{Conf}(C_{k+1}^T) &
\end{tikzcd}$$

Fig. 2. Sequence of rule applications of the TRDM (middle) corresponding to a state transition of the DTM (left). The right column indicates the DTM operation corresponding to each rule.

6 Conclusions

The RDM appears to be an interesting model of computation, inspired by natural processes (chemical, physical) in the same vein of the Chemical Abstract Machine (CHAM) [7]. It has been the basis for the creation of computer based decision support systems and cooperative work systems in environments close to business management [4,5,8], due to its expressiveness and its ability to simulate possible development scenarios and to observe their dynamical evolution in a spatial framework dominated by the local interactions of different entities. This paper is a first step towards the theoretical assessment of their computational power, through the comparison with standard models such as deterministic Turing machines.

References

1. A, Turing. The chemical basis of morphogenesis. *Philos. Trans. R. Society*, 237, 1952.
2. R.S. Cantrell, C. Cosner. Spatial Ecology via Reaction–Diffusion Equations. Wiley, 2003.

3. S. Bandini, G. Mauri, G. Pavesi, C. Simone. A parallel model based on Cellular Automata for the simulation of pesticide percolation in the soil. Proceedings of PACT'99, *Lecture Notes in Computer Science* 1662, 383–394.
4. C. Simone, S. Bandini. The reaction–diffusion methaphor for modeling cooperative work. *Prestige J. of Management and Research*, 2(1): 1–21, 1998.
5. S. Bandini, C. Simone. Integrating forms of interaction in a distributed model. *Fundamenta Informaticae*, 61(1): 1–17, 2004.
6. M.R. Garey, D.S. Johnson, Computers and Intractability. A Guide to the Theory of NP–Completeness. Freeman and Company, San Francisco, 1979.
7. G. Boudol, G. Berry. The chemical abstract machine. *Theoretical Computer Science*, 96(1), 1992.
8. C. Simone, S. Bandini. Integrating awareness in cooperative applications through the reaction–diffusion metaphor. *Computer Supported Cooperative Work* 11(3-4): 495-530, 2002.

Computational Universality in Symbolic Dynamical Systems*

Jean-Charles Delvenne[1], Petr Kůrka[2], and Vincent D. Blondel[1]

[1] Catholic University of Louvain, Department of Mathematical Engineering, Avenue Georges Lemaitre 4, B-1348 Louvain-la-Neuve, Belgium
{delvenne,blondel}@inma.ucl.ac.be

[2] Charles University of Prague, Faculty of Mathematics and Physics, Malostranské náměstí 25, CZ-11800 Praha 1, Czechia
kurka@ms.mff.cuni.cz

Abstract. Many different definitions of computational universality for various types of systems have flourished since Turing's work. In this paper, we propose a general definition of universality that applies to arbitrary discrete time symbolic dynamical systems. For Turing machines and tag systems, our definition coincides with the usual notion of universality. It however yields a new definition for cellular automata and subshifts. Our definition is robust with respect to noise on the initial condition, which is a desirable feature for physical realizability.

We derive necessary conditions for universality. For instance, a universal system must have a sensitive point and a proper subsystem. We conjecture that universal systems have an infinite number of subsystems. We also discuss the thesis that computation should occur at the 'edge of chaos' and we exhibit a universal chaotic system.

1 Introduction

Computability is often defined via universal Turing machines. A Turing machine is a dynamical system, i.e., a set of configurations together with a transformation of this set. Here a configuration is composed of the state of the head and the whole content of the tape. Computation is done by observing the trajectory of an initial point under iterated transformation.

However there is no reason why Turing machines should be the only dynamical systems capable of universal computation, and indeed we know that many systems may perform universal computations.

Artificial neural networks [1], cellular automata [2], billiard balls on a pool table of some complicated form, or a ray of light between a set of mirrors [3] are such systems.

* This paper presents research results of the Belgian Programme on Interuniversity Attraction Poles, initiated by the Belgian Federal Science Policy Office. The scientific responsibility rests with its author(s). J.-C. D. holds a FNRS fellowship (Belgian Fund for Scientific Research). A extended version of this paper, including proofs, is available on http://www.arxiv.org/abs/cs.CC/0404021

For all these systems, many particular definitions of universality have been proposed. Most of them mimic the definition of computation for Turing machines: an initial point is chosen, then we observe the trajectory of this point and see whether it reaches some 'halting' set. See for instance [4] and [5].

However it has been shown that the computational capabilities of many of these systems are strongly affected by the presence of noise [6,7]; fault-tolerant cellular automata are built in [8]. See also [9,10,11] for some definitions of analog computation and issues relative to noise and physical realizability.

Moreover, many variants of these definitions exist and lead to different classes of universal dynamical systems. In particular, there is no consensus for what it means for a cellular automaton to be universal.

Another field of investigation is to make a link between the computational properties of a system and its dynamical properties. For instance, attempts have been made to relate 'universal' cellular automata to Wolfram's classification. It has also been suggested that a 'complex' system must be on the 'edge of chaos': this means that the dynamical behavior of such a system is neither simple (i.e., an attracting fixed point) nor chaotic; see [2,12,13,14]. Other authors nevertheless argue that a universal system may be chaotic: see [1].

The basic questions we would like to address are the following:

– What is a computationally universal dynamical system?
– What are the dynamical properties of a universal system?

A long-term motivation is to answer these questions from the point of view of physics. What natural systems are universal? Is the gravitational N-body problem universal [3]? Are the Navier-Stokes equations universal [15]?

However in this paper we especially focus on *symbolic* dynamical systems, i.e., systems defined on the Cantor set $\{0,1\}^{\mathbb{N}}$ or a subset of it. Some motivating examples of dynamical systems are Turing machines, cellular automata and subshifts. Let us briefly describe our ideas.

Extending Davis' definition of universal Turing machine, we say that a system is universal if some property of its trajectories, such as reachability of the halting set, is r.e.-complete.

However, rather than considering point-to-point or point-to-set properties, we consider set-to-set properties. Typically, given an initial set and a halting set, we look whether there is at least one configuration in the initial set whose trajectory eventually reaches the halting set.

We require the initial and halting sets to be closed open sets of the Cantor space endowed with the usual product topology, which are sets that can be described with a finite number of bits in a natural standard way.

Finally, we do not restrict ourselves to the sole property 'Is there a trajectory going from A to B?' (where A and B are closed open sets), but to any property of closed open sets that can be described in temporal logic.

This definition addresses the two issues raised above. Firstly, it is a general definition directly transposable to any symbolic system. Secondly, dealing with open sets rather than points takes into account some constraints of physical realizability, such as robustness to noise.

With this definition in mind, we prove necessary conditions for a symbolic system to be universal. In particular, we show that a universal symbolic system is not minimal, not equicontinuous and does not satisfy the effective shadowing property. This last property is a variant of the usual shadowing property. We conjecture that a universal system must have infinitely many subsystems, and we show that there is a chaotic system that is universal, contradicting the idea that computation can only happen on the 'edge of chaos'.

The paper is organized as follows: in Section 2 we define effective symbolic systems; in Section 3 the syntax and semantics of temporal logic is exposed; in Section 4 the formal definition of universality is given, and simple examples are provided; this definition is discussed in Section 5; in Section 6 necessary conditions for a system to be universal are given, related to minimality, equicontinuity and effective shadowing property; in Section 7 we build a chaotic system that is universal, and briefly discuss the existence of the 'edge of chaos'; Section 8 discusses possible directions for future work.

2 Effective Symbolic Systems

Effective symbolic dynamical systems are computable continuous transformations of a symbolic space. In this section, we provide a formal definition and elementary examples.

A *symbolic* set is the Cantor set $\{0,1\}^{\mathbb{N}}$ or a subset of it. Some other sets deserve to be called symbolic, for instance $A^{\mathbb{N}}$, $Q \times A^{\mathbb{Z}}$, $A^{\mathbb{Z}^d}$, where A and Q are finite sets and d is a positive integer. But all these sets can be recoded into $\{0,1\}^{\mathbb{N}}$ with standard tricks. Thus, every time we deal with such a set we implicitly suppose that we deal with $\{0,1\}^{\mathbb{N}}$.

Another set of interest is the set of finite and infinite binary words $\{0,1\}^* \cup \{0,1\}^{\mathbb{N}}$. This set can be recoded as a subset of $\{0,1,B\}^{\mathbb{N}}$, if we think of a finite word w as the infinite word $wBBBBBB\dots$ This set can be again recoded as a subset of $\{0,1\}^{\mathbb{N}}$.

The Cantor set can be endowed with the product topology. This topology is given by the metric $d(x,y) = 0$ if $x = y$ and

$$d(x,y) = 2^{-n}$$

where n is the index of the first bit on which x and y differ.

If w is a word of $\{0,1\}^*$, then $[w]$ denotes the set of all sequences beginning by w. In fact, sets of this form, usually called *cylinders*, are exactly the balls of the metric space. Closed open sets (*clopen* sets for short) of $\{0,1\}^{\mathbb{N}}$ are exactly all finite unions of cylinders. Thus clopen sets are finitary objects that can be described by finite words in alphabet $\{0,1\} \cup \{,\}$.

A *symbolic space* is closed subset of $\{0,1\}^{\mathbb{N}}$. It is a topological space for the relative topology, whose clopen sets are the intersections of the closed subset with all clopen sets of Cantor space. A symbolic space is said to be *effective* if checking whether a given clopen set of the Cantor space intersects the symbolic space is decidable.

Definition 1 *An* effective symbolic dynamical system *is a continuous map from an effective symbolic space to itself, such that the inverse map restricted to clopen sets is computable.*

This definition of effective function in a Cantor space is equivalent to classical definitions in computable analysis, for instance [16].

An *effective subsystem* of an effective symbolic system is an effective closed subset that is invariant under the map.

For example, a cellular automaton is an effective symbolic system, acting on the space $A^{\mathbb{Z}^d}$, where A is the finite alphabet and d is the dimension. A Turing machine is an effective system acting on the space $Q \times A^{\mathbb{Z}}$, where Q is the finite set of states of the head and A is the finite tape alphabet.

Recall that a *shift* is a dynamical system on $A^{\mathbb{N}}$ or $A^{\mathbb{Z}}$ (where A is a finite alphabet) with the map $\sigma : A^{\mathbb{N}} \to A^{\mathbb{N}} : a_0a_1a_2a_3 \ldots \mapsto a_1a_2a_3 \ldots$ or $\sigma : A^{\mathbb{Z}} \to A^{\mathbb{Z}} : \ldots a_{-3}a_{-2}a_{-1}\underline{a_0}a_1a_2a_3 \ldots \mapsto \ldots a_{-3}a_{-2}a_{-1}a_0\underline{a_1}a_2a_3 \ldots$, where the symbol of index 0 is underlined. A *subshift* is a subsystem of a shift. A one-sided (two-sided) full shift is an effective system, and if a subshift is an effective closed subset of the Cantor space, then it is again an effective system.

A subshift can be seen as a set of infinite words over a finite alphabet. The set of all finite words appearing at least once in at least one of these words is called the *language* of the subshift. In fact it is easy to see that an effective subshift is exactly a subshift whose language is recursive.

3 Temporal Logic

Our goal is to describe properties of trajectories that may be useful in defining universal computation, such as 'starting from here, the system eventually goes there'. Temporal logic, developed by Prior in 1953, is appropriate to express such properties. It was later used by computer scientists to express and prove sentences such as, typically, 'the program will not reach a forbidden state'; see [17] for a reference book on modal and temporal logic.

Formally, we suppose that we have a set $\{\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, ...\}$ of proposition symbols indexed by $\mathbb{N}$ including two propositions that we will denote $\bot$ and $\top$, and we form all temporal formulae by composing the propositions symbols with the boolean operators $\vee$ and $\neg$, the temporal unary operator $\circ$ (read 'next') and the temporal binary operator $\mathcal{U}$ ('until').

We can also add some usual abbreviations: $\phi \wedge \psi$ denotes $\neg(\neg\phi \vee \neg\psi)$, $\phi \Rightarrow \psi$ denotes $\neg\phi \vee \psi$, $\diamond\phi$ (read 'eventually ϕ') stands for $\top\mathcal{U}\phi$ and $\Box\phi$ (read 'always ϕ') for $\neg \diamond \neg\phi$.

We now give temporal formulae a semantics adapted to symbolic systems. Let (X, f) be an effective symbolic system. Recall that X is a symbolic space and $f : X \to X$ a continuous function. We suppose that clopen sets are numbered in an effective way $P_0, P_1, P_2, \ldots$ Then to each formula ϕ we assign a subset $|\phi|$ of X, called the *interpretation* of ϕ, in the following way.

- If ϕ is the proposition symbol $\mathcal{P}_n$, then $|\phi| = P_n$. Moreover we ask that $|\bot| = \emptyset$ and $|\top| = X$.

– If ϕ is $\phi_1 \vee \phi_2$ then $|\phi| = |\phi_1| \cup |\phi_2|$.
– If ϕ is $\neg\psi$ then $|\phi| = X \setminus |\psi|$.
– If ϕ is $\circ\psi$ then $|\phi| = f^{-1}(|\psi|)$.
– If ϕ is $\phi_1 \mathcal{U} \phi_2$ then $|\phi| = \bigcup_{n\in\mathbb{N}} A_n$, where $A_0 = |\phi_2|$ and $A_{n+1} = f^{-1}(A_n) \cap |\phi_1|$ for all n.

In particular, if ϕ is $\diamond\psi$ then $|\phi| = \bigcup_{n\in\mathbb{N}} f^{-n}(|\psi|)$.

We say that a formula is *satisfiable* if $|\phi| \neq \emptyset$.

Intuitively, we may think that a formula ϕ represents a statement about a point of X, which is seen as 'the current configuration of the system'. This statement may be true for some points of X and false everywhere else. For example, $\diamond\mathcal{P}_n$ means 'when applying f iteratively, the current configuration will eventually be in P_n'. The formula $\mathcal{P}_m\mathcal{U}\mathcal{P}_n$ means 'the configuration lies in P_m until it reaches P_n' or, in other words, 'the configuration will stay in P_m during a finite time and then get in P_n'.

Then $|\phi|$ is the set of points for which the assertion ϕ holds, and a satisfiable formula is verified by at least one configuration. Note that in the following, we will make no distinction between a proposition symbol $\mathcal{P}_n$ and the corresponding clopen set P_n.

4 Universal Systems

We are now ready to state the main definition. We define a universal system to be an effective system with some r.e.-complete temporal property. Then we show that most usual ways to define computability are particular examples of this definition.

Davis [18] proposed the following definition: a Turing machine is universal if the relation 'x_n is in the orbit of x_m' is r.e.-complete, where x_m and x_n are arbitrary finite configurations. Here we modify Davis' definition in order to be applied to any effective symbolic system. Our choices are justified in Section 5.

Definition 2 *An effective dynamical system is* universal *if there is a recursive family of temporal formulae such that knowing whether a given formula of the family is satisfiable is an r.e.-complete problem.*

An *r.e.-complete* problem, or Σ_1-complete problem, is a recursively enumerable problem, to which any recursively enumerable problem is Turing-reducible.

Note that this may be interpreted as a non-deterministic scheme of computation. The computation succeeds iff at least one trajectory exhibits a given behavior.

We may call *halting problem* for f, the satisfiability problem for formulae:

$$(P_n \wedge \diamond P_m)_{n,m\in\mathbb{N}},$$

which reads: 'There is a configuration in the clopen set P_n that eventually reaches the clopen set P_m'. We may think of P_n as an initial configuration of which we know only the first digits and P_m as the halting set. The unspecified digits of the initial configuration may be seen as encoding the non-deterministic choices occurring during the computation.

Turing machines are often described as working only on finite configurations. A finite configuration is an element of $Q \times \{0,1\}^* \times \{0,1\}^*$, where Q denotes the set of states of the head, the first binary word is content of the tape to the left of the head and the second binary word is the right part of the tape. The rest of the tape is supposed to be entirely filled with blank symbols. Such a Turing machine is universal if given two finite configurations u and v, checking whether u is in the trajectory of v is an r.e.-complete problem.

This is a particular case of our definition. Indeed, let $W = \{0,1\}^* \cup \{0,1\}^{\mathbb{N}}$ the set of finite and infinite binary words. Then the Turing machine transition function is also defined on $Q \times W \times W$, which is a compact space, whose isolated points are $Q \times \{0,1\}^* \times \{0,1\}^*$. Isolated points are in fact clopen sets of $Q \times W \times W$. So the problem of checking whether the formula $P_n \wedge \diamond P_m$ is satisfiable, given two clopen sets P_n and P_m, is r.e.-complete. Indeed, it is already r.e.-complete if we restrict ourselves to clopen sets that are isolated points, and it is recursively enumerable (although perhaps not r.e.-complete) on non-isolated clopen sets.

Tag systems were introduced by Post in 1920. A *tag system* is a transformation rule acting on finite binary words. At each step, a fixed number of bits is removed from the beginning of the word and, depending on the values of these bits, a finite word is appended at the end of the word. Minsky proved in 1961 that there is a so-called universal tag system, for which checking that a given word will end up to the empty word when repeating the transformation is an r.e.-complete problem; see [2].

We can extend the rule of tag systems to infinite words, by just removing to them the fixed number of bits. Thus we have a dynamical system on the compact space $\{0,1\}^* \cup \{0,1\}^{\mathbb{N}}$ of finite and infinite words, in which finite words are clopen sets. Again, if the tag system is universal for the word-to-word definition, then it is universal for our definition with the formulae $P_n \wedge \diamond P_m$.

We can also apply our definition to **functions on integers**. Let $\mathbb{N} \cup \{\infty\}$ be the topological space with the metric $d(n,m) = |\frac{1}{n+1} - \frac{1}{m+1}|$. This is effectively homeomorphic to the set $\{1^n 0^\infty | n \in \mathbb{N}\} \cup \{1^\infty\}$. Then a total computable map on $\mathbb{N}$ can be extended to an effective continuous map on $\mathbb{N} \cup \{\infty\}$ iff either it has a finite range and only one integer has an unbounded preimage set, or it has an infinite range and we can compute a (finite) bound on the largest preimage of every given integer.

For example, it is meaningful to ask whether the famous $3n + 1$ function (which is effective) is universal. This is an unsettled question. But Conway [19] proved that similar functions, called Collatz functions, may be universal.

We now give an example of a universal **cellular automaton**.

Let us take a universal Turing machine with a blank symbol. We suppose that when the halting state is reached, then the head comes back to the cell of index 0. We can simulate it in an almost classic way with a one-dimensional cellular automaton. The alphabet of the automaton is $A \cup (A \times Q) \cup \{L, R, Error\}$, where A is the tape alphabet (including the blank symbol) and Q the set of states.

Let us take a point in the cylinder $[L,$ initial data of the Turing machine, $R]$, and observe its trajectory. The symbol L moves to the left at the speed of light, leaving behind blank symbols. The symbol R moves to the right in a similar way. Meanwhile, the space between L and R is used to simulate the Turing machine and is composed of symbols from A and exactly one symbol from $(A \times Q)$, which denotes the position of the head. When L or R symbols meet each other, then a spreading *Error* symbol is produced, that erases everything.

This cellular automaton is universal for formulae $P_n \wedge \diamond P_m$. Indeed, there is an orbit from the cylinder $[L,$ initial data of the Turing machine, $R]$ to the cylinder [halting state] (both cylinders centered at cell of index zero) if and only if the universal Turing machine halts on the initial data.

5 Discussion on the Definition of Universality

Our definition of universality differs in several ways from what we could expect at first glance from a generalization of Turing machine universality. In this section we give various arguments to support the present definition against seemingly more obvious attempts. In particular, we justify the use of *set-to-set* properties, expressed in the formalism of *temporal logic*, on systems for which the transition function is *computable.*

Set-to-Set Properties. Many definitions of universality for particular systems propose to observe point-to-point properties. So it could seem that it is possible to build a general definition of universality with point-to-point properties.

The most natural idea would be to say that a metric space with a dense set of points $(x_n)_{n \in \mathbb{N}}$ is universal if the property 'x_n is in the trajectory of x_m' is r.e.-complete.

However, as remarked in [20], this leads to conclude that the shift is universal; a consequence that is counter-intuitive. It sounds unreasonable to admit the shift as universal, because it does not treat any information, but just reads the memory.

Indeed if instead of ultimately periodic points we choose configurations with primitive recursive digits, then we take as initial configurations the sequence of states of the head of a universal Turing machine during a computation. And we just have to shift to know whether the halting state will appear.

The definition presented in this text overcomes this problem in a simple manner: the user needs only to give a finite number of bits as an initial condition. Instead of initial *configurations* we shall rather talk about initial *sets*, which may be seen as 'fuzzy points', points defined with finite accuracy.

This solution is also more satisfactory from the point of view of physical realizability. Indeed, we expect the set of configurations of a physical system to be uncountable in general, and specifying an initial point for the computation means *a priori* that we must give an infinite amount of information. Preparing a physical system to be in a very particular configuration is likely to be impossible, because of the noise or finite precision inherent to every measure.

Temporal Properties. What kind of property are we going to test on clopen sets? Here again, we must avoid trivialities. Suppose that we look at identity on the Cantor space. We now choose to observe the following property: a clopen set satisfies the property iff its index (i.e., the integer describing the clopen set) satisfies some r.e.-complete property on $\mathbb{N}$. Then we find again that identity is computationally universal, which is desirable.

On the other hand, we see no reason to restrict ourselves to the sole halting property: ‘there is a trajectory from this clopen set to that clopen set’. Any observable property could a priori be used as a basis for computation. For instance, the chaotic system built in Section 7 is universal but not for the halting property.

So we must precisely define a class of observable properties of clopen sets, not too large and not too restricted. Temporal logic, as defined above, has been widely used for decades to express expected properties of various transitions systems and seems to be a reasonable choice.

Effectiveness. Finally, we add an effectiveness structure on dynamical systems, because we want to be able to simulate the system step by step. Indeed, our informal goal is to study when universality emerges from the long-term dynamics. But if even a single step of the system is uncomputable, no surprise that the long-term dynamics is unpredictable!

We therefore restrict ourselves to systems such that the inverse image of a clopen set is computable. Note that for instance in [1] the author allows neural networks with non-recursive weights, leading to a non-computable transition function and to super-Turing capabilities.

6 Necessary Conditions for Universality

It has been highlighted in the Introduction that some attempts have been made to link computational capabilities of a system to its dynamical properties. This is also the purpose of this section.

For simplicity, we will write ‘symbolic system’ for ‘effective symbolic dynamical system’ — unless otherwise specified.

Minimality. A *minimal* dynamical system is a system with no subsystem (except the empty set and itself). It is characterized by the fact that all orbits are dense.

Proposition 1 *A minimal symbolic system is not universal.*

The proof shows by induction that the interpretation of a formula is always a clopen set, and that we can compute it.

Now suppose that the symbolic system is not minimal but consists of several minimal subsystems attracting the whole space of configurations. In other words, the limit set is made of finitely many minimal systems. Recall that the limit set of a dynamical system $f : X \to X$ is the set $\bigcap_{n \geq 0} f^n(X)$.

Proposition 2 *A symbolic system whose limit set is the finite union of minimal subsystems is not universal.*

For example, if all points uniformly converge to a periodic orbit, then the system is not universal. A stronger statement is suggested by the intuition that a universal system is able to simulate many other systems.

Conjecture 1 *A universal symbolic system has infinitely many minimal subsystems.*

Equicontinuity. A system $f : X \to X$ is *equicontinuous* if for all $\epsilon > 0$ there is a $\delta > 0$ such that $d(x, y) < \delta$ implies $d(f^t(x), f^t(y)) < \epsilon$, for any points x, y and nonnegative t.

Proposition 3 *An equicontinuous symbolic system is not universal.*

Again, the proof shows that the interpretation of a formula is a computable clopen set.

We say that a point x of a dynamical system f is *sensitive* if there is an $\epsilon > 0$ such that for every $\delta > 0$ there is a point y with $d(x, y) < \delta$ and a nonnegative time t such that $d(f^t(x), f^t(y)) > \epsilon$.

It is easy to show from compactness that an equicontinuous dynamical system is exactly a system with no sensitive point. Hence, we can deduce from the above result that a universal symbolic system must have a sensitive point.

Equicontinuity in the case of cellular automata has been given a combinatorial characterization in [21]. It is also proved that equicontinuous cellular automata are eventually periodic, thus confirming in this particular case that equicontinuity prevents computational universality from arising.

Shadowing Property. We now define the *effective shadowing property* for a dynamical system.

Definition 3 *Let (X, f) be a dynamical system. A δ-*pseudo-orbit* is a (finite or infinite) sequence of points $(x_n)_{n \geq 0}$ such that $d(f(x_n), x_{n+1}) < \delta$ for all n.*

*A point x ϵ-*shadows* a (finite or infinite) sequence $(x_n)_{n \geq 0}$ if $d(f^n(x), x_n) < \epsilon$ for all n.*

The dynamical system is said to have the shadowing property *if for all $\epsilon > 0$ there is a $\delta > 0$ such that any δ-pseudo-orbit is ϵ-shadowed by some point.*

If moreover such a δ can be effectively computed from ϵ then we say that the system has the effective shadowing property*.*

We can give the following interpretation to this property: suppose that we want to compute numerically the trajectory of x such that at every step numerical errors amount to δ. The resulting sequence of points is a δ-pseudo-orbit, and the shadowing property ensures that this pseudo-orbit is indeed ϵ-close to an actual trajectory of the system.

Proposition 4 *A symbolic system that has the effective shadowing property is not universal.*

The proof shows that a formula is satisfiable iff it is satisfiable for δ-pseudo-orbits, with δ small enough; but the latter property is decidable.

In particular, the full shift is not universal.

The following proposition almost shows that we cannot lift effectiveness of the shadowing property in Proposition 4.

Proposition 5 *There is an undecidable symbolic system that has the shadowing property, but not the effective shadowing property.*

An *undecidable* system is a system for which satisfiability of a given temporal formula is undecidable. We don't know whether this undecidable system is in fact universal.

Note also that Turing machines that satisfy the shadowing property have been given a combinatorial characterization in [22]; in particular, the proof shows that the link between ϵ and δ (see Definition 3) is linear. Hence the effective shadowing property is not stronger than the shadowing property in the case of Turing machines.

7 A Universal Chaotic System and the Edge of Chaos

According to Devaney [23], a system is *chaotic* if it is infinite, topologically transitive and has a dense set of periodic points. We can prove that such a system is sensitive [24].

It is not difficult to prove the existence of a universal subshift. Indeed, consider all the forbidden words of the kind 01^n00^t1, where the universal Turing machine launched on data n does not halt in less than t steps. Then the subshift of all binary sequences avoiding this set of words is effective and universal.

Improving this construction, one gets the following result:

Proposition 6 *There exists an effective system on the Cantor space that is chaotic and universal.*

The central idea of the 'edge of chaos' is that a system that has a complex behavior should be neither too simple nor chaotic. There are several ways to understand that.

Here we interpret 'complex system' by 'universal symbolic system'. Then 'too simple' could refer to the situation treated in Proposition 2: one or several attracting minimal subsystems. This includes of course the case of a globally attracting fixed point.

If we take 'chaotic' as meaning 'Devaney-chaotic', then computational universality need not be on the 'edge of chaos', since we have just provided a chaotic system that is universal.

However, many examples of chaotic systems (whatever the exact meaning given to 'chaotic', and for symbolic systems as well as for analog ones), although not all of them, have the shadowing property and even the effective shadowing property. For instance the shift and Smale's horseshoe (present in some physical systems), as well as all hyperbolic systems, satisfy the effective shadowing property with a linear relation between ϵ and δ (see Definition 3).

Note nevertheless the 'edge of chaos' has been intensively studied for cellular automata. We don't know whether an example of chaotic universal cellular automaton exists.

8 Future Work

Many questions are yet to solve. For instance, we lack sufficient conditions of universality. We didn't investigate in depth what properties a universal cellular automaton satisfies. Let us mention three possible directions for future work.

All formulae involved in examples of universal systems in the preceding sections were quite simple; they were Σ_1 *formulae*, i.e., formulae where no 'until' is negated. We can see that the interpretation of such a formula is a Σ_1 Borel set (an open set) and has a satisfiability problem is recursively enumerable, thus in the level Σ_1 of the arithmetic hierarchy. How far does this correspondence go? Is it true that a universal system is always universal for a family of Σ_1 formulae?

If the symbolic space is endowed with a probability measure (not necessarily invariant for the map), then we would like to check whether a formula is satisfied with positive probability. This can yield a probabilistic definition of universality. How does it relate to the definition developed in this paper?

Can our definition be extended to analog systems? Do the results of Section 6 still apply in this context? For instance, hyperbolic systems are known to have the effective shadowing property. This would suggest that hyperbolic systems are not universal.

References

1. Siegelmann, H.: Neural Networks and Analog Computation: Beyond the Turing Limit. Progress in Theoretical Computer Science. Springer-Verlag (1999)
2. Wolfram, S.: A New Kind of Science. Wolfram Media (2002)
3. Moore, C.: Unpredictability and undecidability in dynamical systems. Physical Review Letters **64** (1990) 2354–2357
4. Siegelmann, H., Fishman, S.: Analog computation with dynamical systems. Physica D **120** (1998) 214–235
5. Bournez, O., Cosnard, M.: On the computational power of dynamical systems and hybrid systems. Theoretical Computer Science **168** (1996) 417–459

6. Asarin, E., Bouajjani, A.: Perturbed Turing machines and hybrid systems. In: Proceedings of the 6th IEEE Symposium on Logic in Computer Science (LICS'01), Boston, USA, IEEE (2001)
7. Maass, W., Orponen, P.: On the effect of analog noise in discrete-time analog computations. Neural Computation **10** (1998) 1071–1095
8. Gacs, P.: Reliable cellular automata with self-organization. In: IEEE Symposium on Foundations of Computer Science. (1997) 90–99
9. Orponen, P.: A survey of continuous-time computation theory. In Du, D.Z., Ko, K.I., eds.: Advances in Algorithms, Languages, and Complexity. Kluwer Academic Publishers (1997) 209–224
10. Moore, C.: Finite-dimensional analog computers: Flows, maps, and recurrent neural networks. In Calude, C., Casti, J., Dinneen, M., eds.: Unconventional Models of Computation, Springer-Verlag (1998)
11. Moore, C.: Dynamical recognizers: real-time language recognition by analog computers. Theoretical Computer Science **201** (1998) 99–136
12. Mitchell, M., Hraber, P., Crutchfield, J.: Dynamic computation, and the "edge of chaos": A re-examination. In Cowan, G., Pines, D., Melzner, D., eds.: Complexity: Metaphors, Models, and Reality. Santa Fe Institute Proceedings, Volume 19, Addison-Wesley (1994) 497–513 Santa Fe Institute Working Paper 93-06-040.
13. Crutchfield, J., Young, K.: Computation at the onset of chaos. In Zurek, W., ed.: Complexity, Entropy and the Physics of Information. Addison-Wesley (1989) 223–269
14. Langton, C.: Computation at the edge of chaos. Physica D **42** (1990) 12–37
15. Moore, C.: Generalized shifts: Unpredictability and undecidability in dynamical systems. Nonlinearity **4** (1991) 199–230
16. Weihrauch, K.: Computable Analysis. Springer-Verlag (2000)
17. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press (2001)
18. Davis, M.: A note on universal Turing machines. In Shannon, C., McCarthy, J., eds.: Automata Studies. Princeton University Press (1956) 167–175
19. Conway, J.: Unpredictable iterations. In: Proceedings of the 1972 Number Theory Conference, Boulder, Colorado (1972) 49–52
20. Durand, B., Róka, Z.: The game of life: universality revisited. In Delorme, M., Mazoyer, J., eds.: Cellular Automata: a Parallel Model. Volume 460 of Mathematics and its Applications. Kluwer Academic Publishers (1999) 51–74
21. Kůrka, P.: Languages, equicontinuity and attractors in cellular automata. Ergodic Theory & Dynamical Systems **17** (1997) 417–433
22. Kůrka, P.: On topological dynamics of Turing machines. Theoretical Computer Science **174** (1997) 203–216
23. Devaney, R.: An Introduction to Chaotic Dynamical Systems. Addison-Wesley (1989)
24. Banks, J., Brooks, J., Cairns, G., Davis, G., Stacey, P.: On Devaney's definition of chaos. American Mathematics Monthly **99** (1992) 332–334

Real Recursive Functions and Real Extensions of Recursive Functions

Olivier Bournez and Emmanuel Hainry

LORIA/INRIA, 615 Rue du Jardin Botanique
54602 Villers-Lès-Nancy, FRANCE
{Olivier.Bournez,Emmanuel.Hainry}@loria.fr

Abstract. Recently, functions over the reals that extend *elementarily* computable functions over the integers have been proved to correspond to the smallest class of real functions containing some basic functions and closed by composition and linear integration.
We extend this result to *all* computable functions: functions over the reals that extend total recursive functions over the integers are proved to correspond to the smallest class of real functions containing some basic functions and closed by composition, linear integration and a very natural unique minimization schema.

1 Introduction

The power of digital discrete time models of computations is rather well understood: all reasonable and sufficiently powerful digital discrete time models have the same power thanks to Turing's work and so-called Church thesis.

For analog models the situation is far from being so clear. Several models have been defined (e.g. the General Purpose Analog Computer (GPAC) model of Shannon [28], neural network models [29,24], hybrid systems [3,4], or theoretical physic models [11,15,23],...) but there are only few results concerning relations between their respective computational power: GPAC computable functions have been characterized mathematically as differentially algebraic functions [12,18,25,28] but this does not provide directly a way to understand the relations between the power of such machines compared to classical discrete machines. Several other analog models have been shown to exhibit super-Turing computational power: using the so-called *Zeno's paradox*, some models make it possible to compute non-Turing computable functions in a constant time: see e.g. [3,5,11,15,19]; the continuity of the space makes it sometimes possible to have models whose power is close to non-uniform complexity classes [29].

Since the progress of electronics and other domains of physics such as mechanics or optics makes the construction of some of the machines realistic, clarifying the situation becomes a crucial matter.

In [19], Moore introduced a class of functions over the reals inspired from the classical characterization of computable functions over integers: observing that the continuous analog of a primitive recursion is a differential equation, Moore proposes to consider the class of $\mathbb{R}$-*recursive functions*, defined as the smallest class of functions containing some basic functions, and closed by composition, differential equation solving (called *integration*), and minimization. The minimization schema of [19] makes it possible to use a *"compression trick"* (another incarnation of Zeno's paradox) to simulate in a bounded time an unbounded number of discrete transitions in order to recognize arithmetical (hence non-Turing-computable) reals [19].

Actually, the original definitions of [19] suffer from several technical problems that appear as soon as the minimization schema is used (see e.g. discussions in [19,9,10,20,21]), and it has been proposed to replace minimization schema by a limit schema to have well-defined classes of functions as in [20,21], or to restrict to functions defined without minimization schema as in [10,12].

Concerning second approach, in his PhD dissertation [10], Campagnolo proposes to consider a class $\mathcal{L}$ of real-functions built in analogy with the class of elementarily computable functions in classical discrete computability: class $\mathcal{L}$ is defined as the smallest class of functions containing some well-chosen basic functions and closed by composition and *linear* integration.

Class $\mathcal{L}$ is proved by Campagnolo *et al.* to be related to functions *elementarily* computable *over the integers* in classical recursion theory: any function over the integers elementary in the sense of classical recursion theory is the restriction to integers of a function that belongs to $\mathcal{L}$ [10,9]; any function in $\mathcal{L}$ that preserves integers has its restriction to integers elementarily computable [10,9].

This paper proves that this is indeed possible to define a reasonable minimization schema to get a class, that we call $\mathcal{L}+!\mu$, that corresponds in a similar way to all (i.e. not necessarily elementary) *computable* functions *over the integers*: we prove that any total recursive function over the integers is the restriction to integers of a function that belongs to $\mathcal{L}+!\mu$, and that any function in $\mathcal{L}+!\mu$ that preserves integers has its restriction to integers total recursive.

Concerning, classical discrete computability, we get a new original characterization of computable functions in terms of restrictions to integers of a natural class of functions over the reals.

Concerning analog models, our results relate the computational power of some algebraically defined classes of functions over the reals to classical discrete models, and hence contribute to understand computations over the reals, or at least to understand the computational power of $\mathbb{R}$-(sub)-recursive functions.

Furthermore the problem we solve is in some sense the definition of a minimization operator, which is strong enough to get at least Turing machine power, but not too strong to get the technical problems of [19], nor non-robust super-Turing Zeno phenomena of [3,5,11,15,19]. In that sense, we believe that our results may be a step toward understanding criteria that could guarantee "robustness" for continuous models as sought by papers like [2,14].

Moreover, we think that that our results could be a first step toward getting an algebraic characterization of functions *over the real numbers* computable *in the sense of recursive analysis*, in the spirit of [6], and alternative to [7,8].

2 Preliminaries

2.1 Mathematical Preliminaries

Let $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{R}$, denote the set of natural integers, the set of rational numbers, and the set of real numbers respectively. Given $x \in \mathbb{R}^n$, we write $\overrightarrow{x}$ to emphasize that x is a vector.

Lemma 1 (Bounding Lemma for Linear Differential Equations (see e.g. [1])). *For linear differential equation $\overrightarrow{x}' = A(t)\overrightarrow{x}$, if A is defined and continuous on interval $I = [a, b]$, where $a \leq 0 \leq b$, then, for all $\overrightarrow{x}_0$, the solution of $\overrightarrow{x}' = A(t)\overrightarrow{x}$ with initial condition $\overrightarrow{x}(0) = \overrightarrow{x}_0$ is defined and unique on I. Furthermore, the solution satisfies $\|\overrightarrow{x}(t)\| \leq \|\overrightarrow{x}_0\| \exp(\sup_{\tau\in[0,t]} \|A(\tau)\|t)$.*

Lemma 2 (Implicit Functions Theorem (see e.g. [26])). *Let $f : \mathbb{R}^{k+1} \to \mathbb{R}$ be a function of class $\mathcal{C}^k$, for $k \geq 1$. Assume that for all $\overrightarrow{x}$, the equation $f(\overrightarrow{x}, y) = 0$ has exactly one solution y. Assume for all $\overrightarrow{x}$ that $\frac{\partial f}{\partial y}(\overrightarrow{x}, y) \neq 0$ in the corresponding root y. Then function $g : \mathbb{R}^k \to \mathbb{R}$ that maps $\overrightarrow{x}$ to the corresponding root y is also of class $\mathcal{C}^k$.*

2.2 Classical Recursion Theory

Classical recursion theory deals with functions over integers. Most classes of classical recursion theory can be characterized as closures of a set of basic functions by a finite number of basic rules to build new functions [27,22]: given a set $\mathcal{F}$ of functions and a set $\mathcal{O}$ of operators on functions (an operator is an operation that maps one or more functions to a new function), $[\mathcal{F}; \mathcal{O}]$ will denote the closure of $\mathcal{F}$ by $\mathcal{O}$.

Proposition 1 (Classical settings: see e.g. [27,22]). *Let f be a function from $\mathbb{N}^k$ to $\mathbb{N}$ for $k \in \mathbb{N}$. Function f is*

- *elementary iff it belongs to $\mathcal{E} = [0, S, U, +, \ominus; \mathrm{COMP}, \mathrm{BSUM}, \mathrm{BPROD}]$;*
- *primitive recursive iff it belongs to $\mathcal{PR} = [0, U, S; \mathrm{COMP}, \mathrm{REC}]$;*
- *total recursive iff it belongs to $\mathcal{R}ec = [0, U, S; \mathrm{COMP}, \mathrm{REC}, \mathrm{MU}]$.*

A function $f : \mathbb{N}^k \to \mathbb{N}^l$ is elementary (resp: primitive recursive, total recursive) iff its projections are elementary (resp: primitive recursive, total recursive).

The basic functions $0, (U_i^m)_{i,m\in\mathbb{N}}, S, +, \ominus$ and the operators BSUM, BPROD, COMP, REC, MU *are given by*

1. $0 : \mathbb{N} \to \mathbb{N}$, $0 : n \mapsto 0$; $U_i^m : \mathbb{N}^m \to \mathbb{N}$, $U_i^m : (n_1, \ldots, n_m) \mapsto n_i$; $S : \mathbb{N} \to \mathbb{N}, S : n \mapsto n + 1$; $+ : \mathbb{N}^2 \to \mathbb{N}$, $+ : (n_1, n_2) \mapsto n_1 + n_2$; $\ominus : \mathbb{N}^2 \to \mathbb{N}$, $\ominus : (n_1, n_2) \mapsto max(0, n_1 - n_2)$;
2. BSUM *: bounded sum. Given f, $h = \mathrm{BSUM}(f)$ is defined by $h : (\overrightarrow{x}, y) \mapsto \sum_{z<y} f(\overrightarrow{x}, z)$;* BPROD *: bounded product. Given f, $h = \mathrm{BPROD}(f)$ is defined by $h : (\overrightarrow{x}, y) \mapsto \prod_{z<y} f(\overrightarrow{x}, z)$;*

3. COMP *: composition. Given* $f_1, \dots, f_p$ *and* g*,* $h = \mathrm{COMP}(f_1, \dots, f_p; g)$ *is defined as the function verifying* $h(\overrightarrow{x}) = g(f_1(\overrightarrow{x}), \dots, f_p(\overrightarrow{x}))$*;*
4. REC *: primitive recursion . Given* f *and* g*,* $h = \mathrm{REC}(f, g)$ *is defined as the function verifying* $h(\overrightarrow{x}, 0) = f(\overrightarrow{x})$ *and* $h(\overrightarrow{x}, n+1) = g(\overrightarrow{x}, n, h(\overrightarrow{x}, n))$*.*
5. MU *: minimization. Given a function* f *such that for all* $\overrightarrow{x}$*, there is a* y *with* $f(\overrightarrow{x}, y) = 0$*, the minimization of* f *is* $\mu f : \overrightarrow{x} \mapsto \inf\{y; f(\overrightarrow{x}, y) = 0\}$*.*

Observe that we consider here only total functions. Furthermore, observe that minimization operator can actually be reinforced into a *unique* minimization operator as follows:

Proposition 2. *A function* f *from* $\mathbb{N}^k$ *to* $\mathbb{N}^l$*, for* $k, l \in \mathbb{N}$*, is total recursive iff its projections belong to* $[0, U, S; \mathrm{COMP}, \mathrm{REC}, \mathrm{UMU}]$ *where operator* UMU *is defined as follows:*

1. UMU*: unique minimization. Given* f *such that for all* $\overrightarrow{x}$ *there is a unique* y *with* $f(\overrightarrow{x}, y) = 0$*, the unique minimization of* f *is defined as the function, denoted by* $!\mu(f)(\overrightarrow{x}, y)$*, that maps* $\overrightarrow{x}$ *to that unique* y*, for all* $\overrightarrow{x}$*.*

Proof. The inclusion $[0, U, S; \mathrm{COMP}, \mathrm{REC}, \mathrm{UMU}] \subset \mathcal{R}ec$ is immediate. Conversely, let ϕ be a function from $\mathcal{R}ec$. It is well known [16,27] that ϕ can be written as $\phi = \chi \circ \mu(\psi)$ with χ and ψ in $\mathcal{E}$ and such that for all $\overrightarrow{x}$, there is at least a y with $\psi(\overrightarrow{x}, y) = 0$ (recall that ϕ is total). Let σ be the elementary function defined by $\sigma(m, n) = \prod_{z<n} \psi(m, z)$. Given m, let us note $n_0 = \mu(\psi)(m)$. We have $\forall n \leq n_0$, $\sigma(m, n) \neq 0$ and $\forall n > n_0$, $\sigma(m, n) = 0$. Let $\kappa(m, n) = 1 \ominus (1 \ominus ((1 \ominus \sigma(m, n)) + \sigma(m, n+1)))$. We have clearly $\forall n < n_0$, $\kappa(m, n) = 1$, $\kappa(m, n_0) = 0$ and $\forall n > n_0$, $\kappa(m, n) = 1$, hence $\mu(\kappa) = !\mu(\kappa) = \mu(\psi)$. κ is an elementary function and we have $\phi = \chi \circ !\mu(\kappa)$, hence ϕ belongs to $[0, U, S; \mathrm{COMP}, \mathrm{REC}, \mathrm{UMU}]$.

We have $\mathcal{E} \subseteq \mathcal{PR} \subseteq \mathcal{R}ec$, and the inclusions are known to be strict [27,22]. If TIME(t) and SPACE(t) denote the classes of functions that are computable with time and space t, then, $\mathcal{PR} = \mathrm{TIME}(\mathcal{PR}) = \mathrm{SPACE}(\mathcal{PR})$ [27,22]. Class $\mathcal{PR}$ corresponds to functions computable using *For-Next programs.* Class $\mathcal{E}$ corresponds to computable functions bounded by some iterate of the exponential function [27,22].

In classical computability, more general objects than functions over the integers can be considered, in particular functionals, i.e. functions $\Phi : (\mathbb{N}^m)^{\mathbb{N}} \times \mathbb{N}^k \to \mathbb{N}^l$. A functional will be said to be *elementary* (or *primitive recursive*, *recursive*) when it belongs to the corresponding[1] class.

[1] Formally, a function f over the integers can be considered as functional $\overline{f} : (V, \overrightarrow{n}) \mapsto f(\overrightarrow{n})$. Similarly, an operator Op on functions $f_1, \dots, f_m$ over the integers can be extended to argument $\overline{Op}(F_1, \dots, F_m) : (V, \overrightarrow{n}) \mapsto Op(f_1(V, .), \dots, f_m(V, .))(\overrightarrow{n})$.

In that spirit, given some set $\mathcal{F}$ of basic functions $\mathbb{N}^k \to \mathbb{N}^l$ and a set $\mathcal{O}$ of operators on functions over the integers, we will still (abusively) denote by $[f_1, \dots, f_p; O_1, \dots, O_q]$ for the smallest class of functionals that contains basic functions $\overline{f_1}, \dots, \overline{f_p}$, plus the functional $Map : (V, n) \to V_n$, the nth element of sequence V, and which is closed by the operators $\overline{O_1}, \dots, \overline{O_q}$. For example, a functional will be said elementary iff it belongs to $\mathcal{E} = [Map, \overline{0}, \overline{S}, \overline{U}, \overline{+}, \overline{\ominus}; \overline{\mathrm{COMP}}, \overline{\mathrm{BSUM}}, \overline{\mathrm{BPROD}}]$.

3 Computable Analysis

The idea sustaining *computable analysis*, also called *recursive analysis*, is to define computable functions over real numbers by considering functionals over fast-converging sequences of rationals [30,17,13,31].

Let $\nu_{\mathbb{Q}} : \mathbb{N} \to \mathbb{Q}$ be the following representation[2] of rational numbers by integers: $\nu_{\mathbb{Q}}(\langle p, r, q\rangle) \mapsto \frac{p-r}{q+1}$, where $\langle ., ., .\rangle : \mathbb{N}^3 \to \mathbb{N}$ is a computable bijection.

A sequence of integers $(x_i) \in \mathbb{N}^{\mathbb{N}}$ *represents a real number* x if $(\nu_{\mathbb{Q}}(x_i))$ converges quickly toward x (denoted by $(x_i) \rightsquigarrow x$) in the following sense : $\forall i, |\nu_{\mathbb{Q}}(x_i) - x| < 2^{-i}$. For $(x_i) \in (\mathbb{N}^k)^{\mathbb{N}}$, we write $(x_i) \rightsquigarrow x$ when it holds componentwise.

Definition 1 (Recursive analysis [31]). *A function $f : \mathbb{R}^k \to \mathbb{R}$ is said computable (or real-computable) if there exists a recursive functional $\Phi : (\mathbb{N}^k)^{\mathbb{N}} \times \mathbb{N} \to \mathbb{N}$ such that for all $\overrightarrow{x} \in \mathbb{R}^k$, for all sequence $X = (\overrightarrow{x}_n) \in (\mathbb{N}^k)^{\mathbb{N}}$, we have $(\phi(X, j))_j \rightsquigarrow f(\overrightarrow{x})$ whenever $X \rightsquigarrow \overrightarrow{x}$. A function $f : \mathbb{R}^k \to \mathbb{R}^l$, with $l > 1$, is said computable if all its projections are.*

A function f will be said *elementarily computable* whenever the corresponding functional Φ is. The class of computable (respectively elementarily computable) functions over the reals will be denoted by $\mathcal{R}ec(\mathbb{R})$ (resp. $\mathcal{E}(\mathbb{R})$).

4 Real-Sub-recursive and Sub-recursive Functions

Following the original ideas from [19], but avoiding the minimization schema of [19] source of many problems, Campagnolo proposed in [10] to consider the following class, built in analogy with elementarily computable functions over the integers.

Definition 2 ([10,9]). *Let $\mathcal{L}$ be the class of functions $f : \mathbb{R}^k \to \mathbb{R}^l$, for some $k, l \in \mathbb{N}$, defined by $\mathcal{L} = [0, 1, -1, \pi, U, \theta_3; \mathrm{COMP}, \mathrm{LI}]$ where the basic functions 0, 1, -1, π, $(U_i^m)_{i,m\in\mathbb{N}}$, θ_3 and the schemata* COMP *and* LI *are the following:*

1. *$0, 1, -1, \pi$ are the corresponding constant functions; $U_i^m : \mathbb{R}^m \to \mathbb{R}$ are, as in the classical settings, projections: $U_i^m : (x_1, \ldots, x_m) \mapsto x_i$;*
2. *$\theta_3 : \mathbb{R} \to \mathbb{R}$ is defined as $\theta_3 : x \mapsto x^3$ if $x \geq 0$, 0 otherwise;*
3. COMP*: composition is defined as in the classical settings: Given $f_1, \ldots, f_p$ and g, $h = \mathrm{COMP}(f_1, \ldots, f_p; g)$ is defined by $h(\overrightarrow{x}) = g(f_1(\overrightarrow{x}), \ldots, f_p(\overrightarrow{x}))$;*
4. LI*: linear integration. From g and h, $\mathrm{LI}(g, h)$ is the maximal solution of the linear differential equation $\frac{\partial f}{\partial y}(\overrightarrow{x}, y) = h(\overrightarrow{x}, y) f(\overrightarrow{x}, y)$ with $f(\overrightarrow{x}, 0) = g(\overrightarrow{x})$.*
   In this schema, if g goes to $\mathbb{R}^n$, $f = \mathrm{LI}(g, h)$ also goes to $\mathbb{R}^n$ and $h(\overrightarrow{x}, y)$ is a $n \times n$ matrix with elements in $\mathcal{L}$.

[2] Many other natural representations of rational numbers can be chosen and provide the same class of computable functions: see [31].

Class $\mathcal{L}$ includes common functions like $+$,sin,cos,$-$,$\times$,exp, or $x \to r$ for all $r \in \mathbb{Q}$ (see [10,9]), but contains only total functions [9]:

Proposition 3 ([9]). *All functions from $\mathcal{L}$ are continuous, defined everywhere, and of class $\mathcal{C}^2$.*

Actually, observing the proofs from [10,9], schema LI can be strengthened as follows:

Proposition 4. *Class $\mathcal{L}$ is also the class of functions $f : \mathbb{R}^k \to \mathbb{R}^l$, for some $k, l \in \mathbb{N}$, defined by $\mathcal{L} = [0, 1, -1, \pi, U, \theta_3; \mathrm{COMP}, \mathrm{CLI}]$ where* CLI *is the following schema:*

1. CLI*: controlled linear integration. From g and h, and c, with h differentiable and entries of h' bounded by c, $\mathrm{CLI}(g, h, c)$ is the maximal solution of the linear differential equation $\frac{\partial f}{\partial y}(\overrightarrow{x}, y) = h(\overrightarrow{x}, y) f(\overrightarrow{x}, y)$ with $f(\overrightarrow{x}, 0) = g(\overrightarrow{x})$. In this schema, if g goes to $\mathbb{R}^n$, $f = \mathrm{CLI}(g, h, c)$ also goes to $\mathbb{R}^n$ and $h(\overrightarrow{x}, y)$ is a $n \times n$ matrix with elements in $\mathcal{L}$.*

Class $\mathcal{L}$ can be related to the class $\mathcal{E}$ of elementarily computable functions over the integers. A *real extension* $\tilde{f}$ of a function $f : \mathbb{N}^k \to \mathbb{N}^l$ over the integers is a function $\tilde{f}$ from $\mathbb{R}^k$ to $\mathbb{R}^l$ whose restriction to $\mathbb{N}^k$ is f. Observe that a function $\tilde{f} : \mathbb{R}^k \to \mathbb{R}^l$ over the reals is an extension of a function over the integers iff its preserves integers: $\tilde{f}(\mathbb{N}^k) \subset \mathbb{N}^l$.

Definition 3 (Discrete Part). *Given a class $\mathcal{C}$ of real functions, we denote by $DP(\mathcal{C})$ the class of functions over the integers that have a real extension in $\mathcal{C}$.*

Proposition 5 ([10,9]). *$\mathcal{E} = DP(\mathcal{L})$. I.e.:*

- *If a function from $\mathcal{L}$ extends some functions over the integers, this latter function is elementarily computable.*
- *Any elementarily computable function over the integers, has a real extension that belongs to $\mathcal{L}$.*

Actually, class $\mathcal{L}$ can also be partially related to the class $\mathcal{E}(\mathbb{R})$ of functions over the real numbers elementarily computable in the sense of recursive analysis: any function from $\mathcal{L}$ is in $\mathcal{E}(\mathbb{R})$ [10,9]. We proved in [6] that the inclusion is actually strict, but that adding a limit schema to class $\mathcal{L}$, allows us to capture whole class $\mathcal{E}(\mathbb{R})$ for functions defined over a compact domain.

5 Real-Recursive and Recursive Functions

We are now going to extend the class $\mathcal{L}$ with a minimization schema in order to get a class whose discrete part correspond to total recursive functions over the integers.

To do so, we need to introduce a zero-finding operator that permits to simulate the classical discrete minimization schema over the integers. However, this operator needs to be stricter than a simple "return the smallest root" since this idea, investigated in [19], has shown to be the source of numerous problems, including ill-defined problems and super-Turing Zeno phenomena [10,9,21,20,19].

Our idea is to use the alternative UMU schema which is equivalent to schema MU for classical computability, but has real counterparts which turn out to preserve real computability.

Indeed, motivated by Proposition 2, by Lemma 2, and by results from recursive analysis about the computability of zeros (see e.g. [31]), we define our unique-zero-finding operator UMU as follows (observe that we also take schema CLI instead of schema LI, which is equivalent when schema UMU is not present):

Definition 4. *Given a differentiable function f from $\mathbb{R}^{k+1}$ to $\mathbb{R}$, if for all $\overrightarrow{x}$, $y \mapsto f(\overrightarrow{x}, y)$ is a non-decreasing function with a unique root y_0, on which $\frac{\partial f}{\partial y}(\overrightarrow{x}, y_0) > 0$, then* UMU$(f)$ *is defined as follows:*

$$\mathrm{UMU}(f) : \begin{cases} \mathbb{R}^k \longrightarrow \mathbb{R} \\ \overrightarrow{x} \;\mapsto\; y_0 \textit{ such that } f(\overrightarrow{x}, y_0) = 0 \end{cases}$$

Let $\mathcal{L}{+}!\mu$ be the set of functions defined by

$$\mathcal{L}{+}!\mu = [0, 1, U, \theta_3; \mathrm{COMP}, \mathrm{CLI}, \mathrm{UMU}].$$

Lemma 3. *$\mathcal{L} \subset \mathcal{L}{+}!\mu$.*

Proof. (sketch) We only need to prove that constant functions -1 and π are in $\mathcal{L}{+}!\mu$. Indeed, -1 is the unique root of $x \mapsto x + 1$, and $\pi = 4\arctan(1)$, where $\arctan(x)$ is the solution of linear differential equation $\arctan(0) = 0$ and $\arctan'(x) = \frac{1}{1+x^2}$, and $x \mapsto \frac{1}{1+x^2}$ can be obtained by applying UMU on $x, y \mapsto (1 + x^2)y - 1$.

Lemma 4. *All functions from $\mathcal{L}{+}!\mu$ are of class $\mathcal{C}^2$ and total.*

Proof. By structural induction. Basic functions 0, 1, U, θ_3 are total and of class $\mathcal{C}^2$. Now, class $\mathcal{C}^2$ and totality are preserved by composition, by linear integration (see e.g. [1]), and by schema UMU by Lemma 2.

Now, observe that operator UMU preserves real computability:

Lemma 5. *Given $f : \mathbb{R}^{k+1} \longrightarrow \mathbb{R}$ real computable, if* UMU(f) *is defined, then* UMU(f) *is also real computable.*

Proof. Given $\overrightarrow{x} \in \mathbb{R}^k$, let y_0 be the unique y_0 with $f(\overrightarrow{x}, y_0) = 0$. Since $f(\overrightarrow{x}, .)$ is continuous, non-decreasing, and with a unique root, we have $f(\overrightarrow{x}, y) < 0$ for $y < y_0$, and $f(\overrightarrow{x}, y) > 0$ for $y > y_0$.

There exists $m \in \mathbb{N}$, such that $f(\overrightarrow{x}, -m) < 0$ and $f(\overrightarrow{x}, m) > 0$: one just need to take any integer m with $-m < y_0 < m$. Actually, such an m can be computed as follows:

```
$m = 1$
Repeat
    Compute $f_1 = f(\overrightarrow{x}, m)$ and $f_2 = f(\overrightarrow{x}, -m)$ at precision $\pm 2^{-m}$
    $m = m + 1$
Until $(f_1 > 2^{-m}$ and $f_2 < -2^{-m})$
Return $m$
```

Indeed, given any integer $m_0 \in \mathbb{N}$ with $-m_0 < y_0 < m_0$, (take for example $\lfloor |y_0| \rfloor + 1$), we have for all $m \geq m_0$, $f(\overrightarrow{x}, m) \geq f(\overrightarrow{x}, m_0) > 0$ and $f(\overrightarrow{x}, -m) \leq f(\overrightarrow{x}, -m_0) < 0$. Now, for m big enough (i.e. $m \geq m_0$, $2^{-m} \leq |f(\overrightarrow{x}, -m_0)|$, and $2^{-m} \leq |f(\overrightarrow{x}, m_0)|$) we have $f_1 > 2^{-m}$ and $f_2 < -2^{-m}$ and the algorithm stops with an m such that $f(\overrightarrow{x}, -m) < 0$ and $f(\overrightarrow{x}, m) > 0$.

Computing y_0 then reduces to compute the unique root of function $f(\overrightarrow{x}, .)$ over a compact $[-m, m]$. The fact that this is indeed computable can be seen as a consequence of the results in [31].

Here is a direct proof: given n, we have to find an approximation of y_0 at precision 2^{-n}. Let us slice $[-m, m]$ in 2^i closed intervals: $[-m, m] = \cup_{0 \leq j < 2^i} [y_j, y_{j+1}]$ where $y_j = -m + j\frac{2m}{2^i}$. Let z_j be an approximation of $f(\overrightarrow{x}, y_j)$ computed at precision 2^{-i}. We know that for a root to exist in $[y_j, y_{j+1}]$, the only possibilities are that $|z_j| < 2^{-i}$ or $|z_{j+1}| < 2^{-i}$ or $z_j z_{j+1} < 0$[3]. Then, let m_i be the y_j (resp. M_i be the y_{j+1}) where index j is the smallest (resp. greatest) integer $0 \leq j < 2^i$ with $|z_j| < 2^{-i}$ or $|z_{j+1}| < 2^{-i}$ or $z_j z_{j+1} < 0$.

The sequences (m_i) and (M_i) have range in compact sets, so there exist subsequences $(m_{\phi(i)})$ and $(M_{\phi(i)})$ that converge, thanks to Bolzano-Weierstrass theorem. Let m^* and M^* be the limits of those sequences. For all i, either $|f(\overrightarrow{x}, m_i)| \leq |f(\overrightarrow{x}, m_i) - z_j| + |z_j| < 2^{-i} + 2^{-i}$, or $|f(\overrightarrow{x}, m_i + 2^{-i})| \leq |f(\overrightarrow{x}, m_i) - z_{j+1}| + |z_{j+1}| < 2^{-i} + 2^{-i}$, or $f(\overrightarrow{x}, m_i) f(\overrightarrow{x}, m_i + 2^{-i}) < 0$. Since f is continuous, we can deduce that $f(\overrightarrow{x}, m^*) = 0$. For the same reason, $f(\overrightarrow{x}, M^*) = 0$ and since $y \mapsto f(\overrightarrow{x}, y)$ has only one root, $m^* = M^*$. So, there exists i such that $M_i - m_i < 2^{-n}$. When this holds, m_i is an approximation at precision 2^{-n} of the root. This means that the following algorithm terminates and returns an approximation of y_0 at precision 2^{-n}.

```
$i = 0$
Repeat
    Compute $m_i$ and $M_i$
    $i = i + 1$
Until $M_i - m_i < 2^{-n}$
Return $m_i$
```

Lemma 6. *Given h, g and c real computable, then $f = \mathrm{CLI}(g, h, c)$ is also real computable.*

[3] In fact, since the function we are investigating is non-decreasing, we could have more accurate constraints, however these ones are sufficient.

Proof. Observing carefully [10,9], if given $\overrightarrow{x} \in \mathbb{R}^k$ and some $\overline{y} \in \mathbb{Q}$ one can bound effectively the norms of $h(\overrightarrow{x}, y)$, $f(\overrightarrow{x}, y)$, $\frac{\partial^2 f}{\partial y^2}(\overrightarrow{x}, y)$ for $|y| \leq \overline{y}$, then f will be real computable: use the constructions and bounds based on Euler's method to prove preservation of elementarily computability by linear integration in [10,9], but replacing elementary bounds by computable bounds.

Now, from [31], it is known that one can bound effectively the norm of any real computable function on a compact domain, and so we only need to care about $f(\overrightarrow{x}, y)$ and $\frac{\partial^2 f}{\partial y^2}(\overrightarrow{x}, y)$. But the norm of $f(\overrightarrow{x}, y)$ can be bounded effectively by Lemma 1 from bounds on the norms of $g(\overrightarrow{x})$ and $h(\overrightarrow{x}, y)$ on the corresponding domain, which are computable by previous argument. Now, $\|\frac{\partial^2 f}{\partial y^2}(\overrightarrow{x}, y)\| = \|(h^2(\overrightarrow{x}, y)+\frac{\partial h}{\partial y}(\overrightarrow{x}, y))f(\overrightarrow{x}, y)\|$, hence is bounded by $(\|h^2(\overrightarrow{x}, y)\|+\|c(\overrightarrow{x}, y))\|) \times \|f(\overrightarrow{x}, y)\|$. First factor can still be bounded effectively since $h^2(\overrightarrow{x}, y)$ and $c(\overrightarrow{x}, y)$ are particular real computable functions, and we just see that second factor can be bounded effectively.

From previous two Lemmas, the fact that basic functions are real computable and observing that composition is known to preserve real computability for total functions (see [31]), we obtain:

Theorem 1. *Every function belonging to $\mathcal{L}+!\mu$ is real computable.*

We now prove the converse direction. Following lemma is a weaker form of a Lemma that we proved in [6]:

Lemma 7. *Given $f : \mathbb{R}^2 \to \mathbb{R}$ in $\mathcal{L}$, there exists $\tilde{f} : \mathbb{R}^2 \to \mathbb{R}$ in $\mathcal{L}$ such that $\forall (m, n) \in \mathbb{N}^2$, $\forall (x, y) \in \mathbb{R}^2$,*

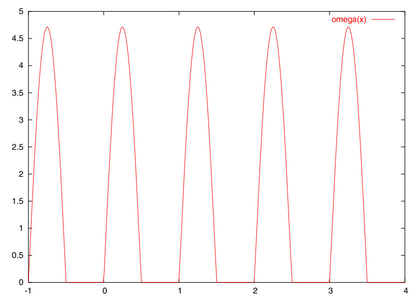
- $\tilde{f}(m, n) = f(m, n)$
- $\tilde{f}(m, y) \in [f(m, \lfloor y \rfloor), f(m, \lfloor y + 1 \rfloor)]$ *(or* $[f(m, \lfloor y + 1 \rfloor), f(m, \lfloor y \rfloor)]$*).*
- $\tilde{f}(x, n) \in [f(\lfloor x \rfloor, n), f(\lfloor x + 1 \rfloor, n)]$ *(or* $[f(\lfloor x + 1 \rfloor, n), f(\lfloor x \rfloor, n)]$*).*

Proof. Let $\zeta = \frac{3\pi}{2}$. Let $\omega : x \mapsto \zeta\theta_3(\sin(2\pi x))$. $\forall i, \int_i^{i+1} \omega = 1$ and ω is equal to 0 on $[i+\frac{1}{2}, i+1]$ for $i \in \mathbb{N}$. Let Ω its primitive equal to 0 in 0, and $int : x \mapsto \Omega(x-\frac{1}{2})$. Function int is a function similar to the integer part: $\forall i \in \mathbb{N}$, $\forall x \in [i, i + \frac{1}{2}]$, $int(x) = i = \lfloor x \rfloor$. Figure 1 shows graphical representations of ω and int.

Let $\Delta(i, y) = f(i, y + 1) - f(i, y)$. Then for all $i \in \mathbb{N}$, $y \in \mathbb{R}$, we have $\omega(y)\Delta(i, int(y)) = \begin{cases} 0 & \text{whenever } y - \lfloor y \rfloor \geq 1/2 \\ \omega(y)\Delta(i, \lfloor y \rfloor) & \text{otherwise.} \end{cases}$

Let G be the solution of the linear differential equation $G(x, 0) = f(x, 0)$, $\frac{\partial G}{\partial y}(x, y) = \omega(y)\Delta(x, int(y))$. An easy induction on j then shows that $G(i, j) = f(i, j)$ for all integer j. Furthermore, by construction, $\forall i \in \mathbb{N}$, $G(i, y)$ belongs to the interval delimited by $G(i, \lfloor y \rfloor) = f(i, \lfloor y \rfloor)$ and $G(i, \lfloor y + 1 \rfloor) = f(i, \lfloor y + 1 \rfloor)$.

Now, let $\tilde{f}$ be the solution of the linear differential equation $\tilde{f}(0, j) = G(0, j)$, $\frac{\partial \tilde{f}}{\partial x}(x, y) = \omega(x)(G(int(x + 1), y) - G(int(x), y))$. We have $\forall (i, j) \in \mathbb{N}^2$, $\tilde{f}(i, j) = f(i, j)$. And $\forall i \in \mathbb{N}$, $\tilde{f}(i, y)$ belongs to the interval delimited by $\tilde{f}(i, \lfloor y \rfloor) = f(i, \lfloor y \rfloor)$ and $\tilde{f}(i, \lfloor y + 1 \rfloor) = f(i, \lfloor y + 1 \rfloor)$. And also, $\forall j \in \mathbb{N}$, $\tilde{f}(x, j)$ belongs to the interval delimited by $\tilde{f}(\lfloor x \rfloor, j) = f(\lfloor x \rfloor, j)$ and $\tilde{f}(\lfloor x + 1 \rfloor, j) = f(\lfloor x + 1 \rfloor, j)$.



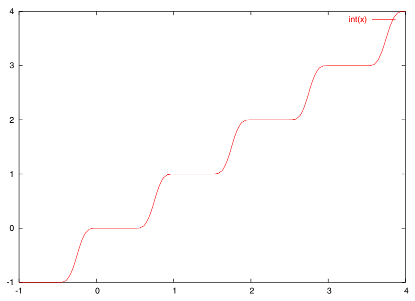


Fig. 1. Graphical representation of ω and *int*

Theorem 2. *Every recursive function over the integers has a real extension in* $\mathcal{L}+!\mu$.

Proof. Let ϕ be a function from $\mathcal{R}ec$. We have $\phi = \chi \circ !\mu(\kappa)$ as in the proof of Proposition 2. Let $\iota(m,n) = 2 \times (1 \ominus \sigma(m,n)) + (1 \ominus \kappa(m,n))$ where σ is the same as in the proof of Proposition 2. $\forall m \in \mathbb{N}$, for $n = n_0 = !\mu(\kappa)(m,n)$, we have $\iota(m,n_0) = 1$, and before this n_0, $\iota(m,n)$ is equal to 0 and after this n_0, $\iota(m,n)$ is equal to 2. Let i be a real extension of ι in $\mathcal{L}$ given by Proposition 5. Let $\tilde{i}$ be the function from $\mathcal{L}$ obtained by Lemma 7 on $f(m,x) : m,x \mapsto i(m,x) - 1$.

$\forall m \in \mathbb{N}$, there exists exactly one $y \in \mathbb{R}$ (given by $y_0 = !\mu(\kappa)(m,n)$) such that $\tilde{i}(m,y) = 0$. But, we can not directly apply schema UMU, since we have no assurance[4] that it also holds for non integer values m. However, from the constructions in the proof of Lemma 7, given $m \in \mathbb{N}$, we have $\tilde{i}(m,y)$ equal to -1 for $y \leq y_0 - 1$, and equal to $\Omega(y)$ for $y \in [y_0 - 1, y_0 + 1]$, where Ω is defined in that proof.

Consider $\mathcal{M}(x) = \theta_3(x+1)$. We have $\mathcal{M}(x) = 0$ if $x \leq -1$ and $\mathcal{M}(x) \geq 1$ if $x \geq 0$. Let us define $\tilde{g}$ as the solution of the differential equation $\tilde{g}(\overrightarrow{x}, 0) = -1$, $\frac{\partial \tilde{g}}{\partial y}(\overrightarrow{x}, y) = \alpha \mathcal{M}(\tilde{i}(\overrightarrow{x}, y))$. Let us choose α (maple says $\alpha = \frac{1024}{2609}$) such that $\alpha \int_{-1}^{0} \mathcal{M}(\Omega(x)) \mathrm{d}x = 1$. We have $\forall m \in \mathbb{N}$, $\tilde{g}(m,y) = 0 \Leftrightarrow y = !\mu(\kappa)(m,n)$.

Then define g as the solution of the linear differential equation $g(\overrightarrow{x}, 0) = -1$, $\frac{\partial g}{\partial y}(\overrightarrow{x}, y) = \beta \mathcal{M}(\tilde{g}(\overrightarrow{x}, y))$. If we choose β adequately[5] (maple says $\beta = \frac{a\pi^4}{b\pi^4 - c\pi^2 + d}$ for some integers a,b,c,d) , we will still have $\forall m \in \mathbb{N}$, $g(m,y) = 0 \Leftrightarrow y = !\mu(\kappa)(m,n)$.

The point is that, since $\mathcal{M}$ is always non-negative, we know that $\forall x \in \mathbb{R}$, $y \mapsto \tilde{g}(x,y)$ is non-decreasing, and, because of Lemma 7, and from the definition of function $\mathcal{M}(x)$, it must go to infinity when y goes to infinity. Actually, it must be equal to -1 up to a certain value y_-, then be strictly increasing, and since it goes to infinity, it must have a root y_0 strictly greater than y_-. Now the derivative in this root y_0 cannot be 0 since $\mathcal{M}(x)$ is zero only when $x \leq -1$.

[4] Actually, another problem is that the derivative relative to the second variable in the root point is 0.

[5] This β is in $\mathcal{L}$ since it can be obtained as $a * \pi^4 * \mathrm{UMU}(x \mapsto (b\pi^4 - c\pi^2 + d)x - 1)$.

This g is such that $\forall \overrightarrow{x}$, $\exists! y_0$ such that $g(\overrightarrow{x}, y_0) = 0$ and $\frac{\partial g}{\partial y}(\overrightarrow{x}, y_0) \neq 0$ and for all $\overrightarrow{x}$, $y \mapsto g(\overrightarrow{x}, y)$ is non-decreasing. We can thus apply UMU to this g. Now if we extend χ in a real function h belonging to $\mathcal{L}$ using Proposition 5, we have $h \circ \mathrm{UMU}(g)$ extending $\phi = \chi \circ \mu(\psi)$ and belonging to $\mathcal{L}+!\mu$.

From previous two theorems, we obtain the main result of this paper:

Theorem 3. *$\mathcal{R}ec = DP(\mathcal{L}+!\mu)$. I.e:*

- *If a function from $\mathcal{L}+!\mu$ extends some function over the integers, this latter function is total recursive.*
- *Any total recursive function over the integers, has a real extension that belongs to $\mathcal{L}+!\mu$.*

Proof. The second item is Theorem 2. The first item is immediate from Theorem 1: if a function f belonging to $\mathcal{L}+!\mu$ preserves integers, then a recursive function that equals f on $\mathbb{N}^k$ can easily be obtained from the functional computing f.

Corollary 1. *$\mathcal{L}$ is strictly included in $\mathcal{L}+!\mu$.*

References

1. V. I. Arnold. *Ordinary Differential Equations.* MIT Press, 1978.
2. E. Asarin and A. Bouajjani. Perturbed Turing machines and hybrid systems. In *Logic in Computer Science*, pages 269–278, 2001.
3. E. Asarin and O. Maler. Achilles and the tortoise climbing up the arithmetical hierarchy. *Journal of Computer and System Sciences*, 57(3):389–398, dec 1998.
4. O. Bournez. Achilles and the Tortoise climbing up the hyper-arithmetical hierarchy. *Theoretical Computer Science*, 210(1):21–71, 6 1999.
5. O. Bournez. *Complexité Algorithmique des Systèmes Dynamiques Continus et Hybrides.* PhD thesis, Ecole Normale Supérieure de Lyon, Janvier 1999.
6. O. Bournez and E. Hainry. An analog characterization of elementary computable functions over the real numbers. In *International Colloquium on Automata, Languages and Programming (ICALP 2004)*, volume 3142 of *Lecture Notes in Computer Science*, pages 269–280, 2004.
7. V. Brattka. Recursive characterizations of computable real-value functions and relations. *Theoretical Computer Science*, 162(1):45–77, 5 August 1996.
8. V. Brattka. Computability over topological structures. In S. B. Cooper and S. S. Goncharov, editors, *Computability and Models*, pages 93–136. Kluwer Academic Publishers, New York, 2003.
9. M. Campagnolo, C. Moore, and J. F. Costa. An analog characterization of the Grzegorczyk hierarchy. *Journal of Complexity*, 18(4):977–1000, 2002.
10. M. L. Campagnolo. *Computational complexity of real valued recursive functions and analog circuits.* PhD thesis, Universidade Técnica de Lisboa, 2001.
11. G. Etesi and I. Németi. Non-Turing computations via Malament-Hogarth space-times. *International Journal Theoretical Physics*, 41:341–370, 2002.
12. D. Graça and J. F. Costa. Analog computers and recursive functions over the reals. *Journal of Complexity*, 19:644–664, 2003.

13. A. Grzegorczyk. Computable functionals. *Fundamenta Mathematicae*, 42:168–202, 1955.
14. T. Henzinger and J.-F. Raskin. Robust undecidability of timed and hybrid systems. *Hybrid Systems: Computation and Control; Second International Workshop, HSCC'99, Berg en Dal, The Netherlands, march 29–31, 1999; proceedings*, 1569, 1999.
15. M. L. Hogarth. Does general relativity allow an observer to view an eternity in a finite time? *Foundations of Physics Letters*, 5:173–181, 1992.
16. L. Kalmár. Egyszerü példa eldönthetetlen aritmetikai problémára. *Mate és fizikai lapok*, 50:1–23, 1943.
17. D. Lacombe. Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles III. *Comptes rendus de l'Académie des Sciences Paris*, 241:151–153, 1955.
18. L. Lipshitz and L. A. Rubel. A differentially algebraic replacement theorem, and analog computability. *Proceedings of the American Mathematical Society*, 99(2):367–372, February 1987.
19. C. Moore. Recursion theory on the reals and continuous-time computation. *Theoretical Computer Science*, 162(1):23–44, 5 1996.
20. J. Mycka. Infinite limits and R-recursive functions. *Acta Cybernetica*, 16:83–91, 2003.
21. J. Mycka. μ-recursion and infinite limits. *Theoretical Computer Science*, 302:123–133, 2003.
22. P. Odifreddi. *Classical recursion theory II.* North-Holland, 1999.
23. T. Ord. Hypercomputation: computing more than the Turing machine. Technical report, University of Melbourne, september 2002. See http://www.arxiv.org/abs/math.lo/0209332.
24. P. Orponen. A survey of continuous-time computation theory. In D.-Z. Du and K.-I. Ko, editors, *Advances in Algorithms, Languages, and Complexity*, pages 209–224. Kluwer Academic Publishers, Dordrecht, 1997.
25. M. B. Pour-El. Abstract computability and its relation to the general purpose analog computer (some connections between logic, differential equations and analog computers). *Transactions of the American Mathematical Society*, 199:1–28, 1974.
26. E. Ramis, C. Deschamp, and J. Odoux. *Cours de mathématiques spéciales, tome 3, topologie et éléments d'analyse.* Masson, feb 1995.
27. H. Rose. *Subrecursion: functions and hierarchies.* Clarendon Press, 1984.
28. C. E. Shannon. Mathematical theory of the differential analyser. *Journal of Mathematics and Physics MIT*, 20:337–354, 1941.
29. H. Siegelmann. *Neural networks and analog computation - beyond the Turing limit.* Birkauser, 1998.
30. A. Turing. On computable numbers, with an application to the "Entscheidungsproblem". In *Proceedings of the London Mathematical Society*, volume 2, pages 230–265, 1936.
31. K. Weihrauch. *Computable analysis.* Springer, 2000.
32. Q. Zhou. Subclasses of computable real valued functions. *Lecture Notes in Computer Science*, 1276:156–165, 1997.

Ordering and Convex Polyominoes

Giusi Castiglione and Antonio Restivo

University of Palermo, Dipartimento di Matematica e Applicazioni,
Via Archirafi 34, 90123 Palermo, Italy
{giusi,restivo}@math.unipa.it

Abstract. We introduce a partial order on pictures (matrices), denoted by $\preceq$ that extends to two dimensions the subword ordering on words. We investigate properties of special families of discrete sets (corresponding to $\{0,1\}$-matrices) with respect to this partial order. In particular we consider the families of polyominoes and convex polyominoes and the family, recently introduced by the authors, of L-convex polyominoes.
In the first part of the paper we study the closure properties of such families with respect to the order. In particular we obtain a new characterization of L-convex polyominoes: a discrete set P is a L-convex polyomino if and only if all the elements $Q \preceq P$ are polyominoes.
In the second part of the paper we investigate whether the partial orderings introduced are well-orderings. Since our order extends the subword ordering, which is a well-ordering (Higman's theorem), the problem is whether there exists some extension of Higman's theorem to two dimensions. A negative answer is given in the general case, and also if we restrict ourselves to polyominoes and even to convex polyominoes. However we prove that the restriction to the family of L-convex polyominoes is a well-ordering. This is a further result that shows the interest of the notion of L-convex polyomino.

1 Introduction

In the study of computational models working on two-dimensional grids, combinatorial properties of picture play an important role. In this paper we study properties of significant families of matrices (pictures) over a finite alphabet with respect to the following order relation: given two matrices P and Q , we say that $P \preceq$ iff P can be obtained from Q by deleting some rows and/or columns. This is a very natural order relation on matrices: it generalizes to two dimensions the notion of (scattered) subword of a word, a notion that has been largely investigated in combinatorics on words and in formal language theory (cf. [13]). Moreover, such a partial ordering has been considered in [14] for the definition and the study of some special families of picture languages. More generally, deletion (and insertion) are fundamental operations in computational processes in words and pictures.

Discrete sets, i.e. finite subsets of $\mathbb{Z}^2$, correspond to $\{0,1\}$-matrices. The most frequently used properties of discrete sets are connectedness and convexity. Such properties lead to the notions of polyominoes and convex polyominoes, that have been extensively investigated both from combinatorial and the algorithmic point of view (cf. [1], [2], [12]).

Recently, a special subclass of convex polyominoes has been introduced by the authors. Their elements are called *L-convex Polyominoes* and they have striking properties as to concern both their reconstruction and their enumeration (cf. [3], [4]).

This paper investigates properties of different classes of discrete sets, with respect to the partial ordering given above. In particular, in Sec. 3.2 we study the *down-sets* corresponding to the classes of polyominoes, convex polyominoes and L-convex polyominoes. As a main result of this section, we obtain a purely order-theoretic characterization of L-convex polyominoes: a polyomino P is L-convex if and only if, for any discrete set Q such that $Q \preceq P$, one has that Q is a polyomino too.

An important notion on partial ordering, from the viewpoint of our considerations, is that of *well-ordering*, i.e. a partial ordering in which every set of pairwise incomparable elements is finite. There exist various characterizations of this concept which was often rediscovered by different authors (cf. [9], [11]). Well quasi-orders are important in mathematics and in many areas of theoretical computer science as well. A basic theorem in such a theory is Higman's theorem, stating, in particular that, if we consider the set of words over a finite alphabet, the subword partial order is a well ordering. Higman's theorem has been extended to structures more general than words as, for instance, labelled trees (cf. [10]) and infinite words (cf. [7]). A natural question is whether such a theorem can be extended to two-dimensional words (matrices). A negative answer is given here. Actually, in Sec. 4, we prove a stronger result: the class of convex polyominoes, with the partial order here considered, is not a well-ordering. However, as main result, of the section, we prove that the class of L-convex polyominoes is a well-ordering. Such a theorem in one hand provides a non trivial generalization of Higman's theorem to an important class of bidimensional objects, on the other hand it shows more and on the interest of the notion of L-convex polyomino.

2 Preliminaries

In this section we give basic definitions about the class of discrete sets and polyominoes, we introduce L-convex polyominoes and their properties.

2.1 Discrete Sets and Polyominoes

A *discrete set* P is a finite subset of the lattice $\mathbb{Z}^2$ defined up to translation. We denote by $\mathcal{D}$ the class of discrete sets. Let $P \in \mathcal{D}$, and $m \times n$ be the size of the minimal bounding rectangle, so P can be represented as a binary matrix $(P_{ij})_{m\times n}$ such that

$$P_{ij} = \begin{cases} 1 & \text{if } (i,j) \in P \\ 0 & \text{otherwise} \end{cases}$$

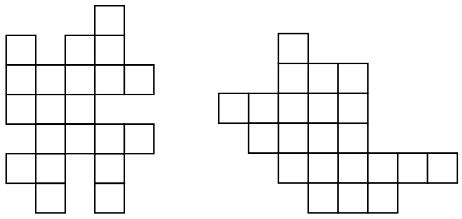


Fig. 1. Polyomino and convex polyomino.

In what follows we denote by (i, j) a generic element of the lattice and by $P(i, j)$ a generic element of P (i.e. such that $P_{ij} = 1$) that we call *cell* of P.

Two cell $P(i, j)$ and $P(i', j')$ are *adjacent* if ($i' = i \pm 1$ and $j' = j$) or ($i' = i$ and $j' = j \pm 1$). A *path*, Π_{AB}, connecting two cells A and B, is a sequence

$$(A = (i_1, j_1), (i_2, j_2), ..., (i_r, j_r) = B)$$

of adjacent cells. The step $((i_k, j_k), (i_{k+1}, j_{k+1}))$ is called:

– an *Est* step if $i_{k+1} = i_k + 1$ and $j_{k+1} = j_k$;
– a *North* step if $i_{k+1} = h_i$ and $j_{k+1} = j_k + 1$;
– a *West* step if $i_{k+1} = h_i - 1$ and $j_{k+1} = j_k$;
– a *South* step if $i_{k+1} = h_i$ and $j_{k+1} = j_k - 1$.

We say that a path is *monotone* if it is made with steps in only two directions. Two cells are *connected* if they can be connected by a path.

A particular class of discrete sets is the class of *polyominoes*. A *polyomino* is a discrete set in which every cell is connected to each other. The class of polyominoes is here denoted by $\mathcal{P}$. A polyomino is said to be *h-convex* (resp. *v-convex*) if every its row (resp. column) is connected. A polyomino is said to be *hv-convex*, or simply *convex*, if it is both h-convex and v-convex (see Fig. 1). We will denote by $\mathcal{C}$ the class of convex polyominoes.

If Q is a polyomino and we consider a subset of cells with the same property of connection, we obtain another polyomino $P \subseteq Q$. Since a polyomino is defined up to translation we can have two distinct subsets of cells of Q that determines the same polyomino P. In this case we say that P has two occurrences in Q. In any case, we say that P is included in Q and we write $P \subseteq Q$.

2.2 L-convex Polyominoes

Cells of convex polyominoes satisfy a particular connection property stated in the following Proposition 1 (cf. [3]). Such a result allows to introduce a particular family of convex polyominoes, called *L-convex*, defined and studied in [3].

Proposition 1. *A polyomino P is convex iff every pair of cells is connected by a monotone path.*

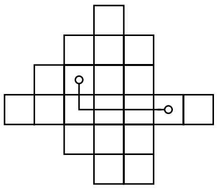


Fig. 2. L-convex Polyomino.

A path has a *change of direction* in the cell (i_k, j_k), for $2 \leq k \leq r-1$, if

$$i_k \neq i_{k-1} \Longleftrightarrow j_{k+1} \neq j_k.$$

Taking into account maximal number of change of direction in their monotone paths, we have a classification of convex polyominoes. In particular, we call *k-convex* a convex polyomino such that every pair of cells can be connected by a monotone path with at most k changes of direction. Then, at first level of this classification we have *1-convex* polyominoes called *L-convex* polyominoes.

Definition 1. *An L-convex polyomino P is a convex polyomino in which every pair of cell is connected by a path with at most one change of direction, for its shape called* L-path*(see Fig.2).*

We denote by $\mathcal{L}$ the class of L-convex polyominoes. It is easy to prove following lemma.

Lemma 1. *Let $P \in \mathcal{C}$, P is L-convex if and only if*

$$\forall\ (i,j),(h,k) \in P \Rightarrow (i,k) \in P \ \textit{or}\ (h,j) \in P.$$

There is an important characterization of L-convex polyominoes that takes into account the position of rectangles that they include.

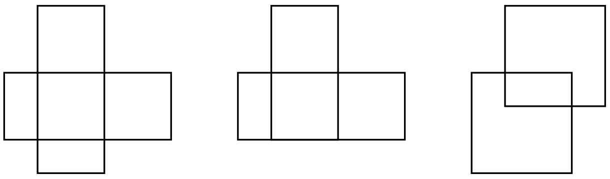


Fig. 3. The first two are examples of crossing intersection and the third one is an example of non crossing intersection.

A *rectangle*, that we denote by $[x,y]$, with $x,y \in \mathbb{N} \setminus \{0\}$, is a rectangular polyomino whose dimensions are x and y, respectively. We denote by $\mathcal{R}$ the set of rectangles. Given a convex polyomino P, we denote by $\mathcal{R}(P) = \{[x,y]/[x,y] \subseteq P\}$. We say $[x,y]$ to be *maximal* in P if

$$\forall\, [x',y'],\ [x,y] \subseteq [x',y'] \subseteq P \ \Rightarrow\ [x,y] = [x',y']$$

We denote by $\mathcal{R}_{max}(P)$ the set of the maximal elements of $\mathcal{R}(P)$.

Given two occurrences of the rectangles $[x,y]$ and $[x'y']$ in a polyomino P, respectively, we say that they have a *crossing intersection*, if their intersection is a rectangle with basis the smallest of two basis and height the smallest of two heights. See Fig.3 for example.
The following theorem has been proved in [3].

Theorem 1. *A convex polyomino P is L-convex iff every pair of its maximal rectangles occurs in P with a crossing intersection.*

Since two different occurrences of the same maximal rectangle $[x,y]$ in P should have crossing intersection, as consequence of Theorem 1 we have that each maximal rectangle of a L-convex polyomino P has a *unique* occurrence in P.
In this way we can describe a L-convex polyomino as a finite overlapping of not comparable rectangles such that any pair of them has a crossing intersection (see Fig.4 for example).

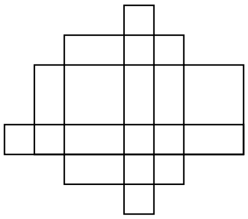


Fig. 4. L-convex polyomino with four maximal rectangles.

If P is an L-convex polyomino and $[x,y] \in \mathcal{R}_{max}(P)$, with the notation $P \setminus [x,y]$ we mean the unique L-convex polyomino included in P such that $\mathcal{R}_{max}(P \setminus [x,y]) = \mathcal{R}_{max}(P) \setminus \{[x,y]\}$.

3 An Order Relation

Let $\mathcal{M}$ be the set of matrices (or picture) over a finite alphabet. In (cf.[14]) a binary relation on $\mathcal{M}$ is given as follows.

Definition 2. *Let $P, Q \in \mathcal{M}$ such that P has dimension $m \times n$. Then $\preceq Q$ (or P is* subpicture*of Q) if there are strictly monotone functions $r : \{1, ..., n\} \rightarrow \mathbb{N}_{\geq 1}$ and $c : \{1, ..., m\} \rightarrow \mathbb{N}_{\geq 1}$ such that $P_{ij} = Q_{r(i)c(j)}$ for all $(i, j) \in \{1, ..., n\} \times \{1, ..., m\}$.*

This binary relation is, trivially, transitive, reflexive and antisymmetric then $(\mathcal{M}, \preceq)$ is an *ordering.* We say that P and Q are *comparable* if either $P \preceq Q$ or $Q \preceq P$; otherwise we say that they are *incomparable.*

Example 1. P and Q are two matrices over the finite alphabet $\Sigma = \{a, b, c\}$.

$$P = \begin{pmatrix} a & b & c \\ a & b & b \\ c & b & b \end{pmatrix} \quad Q = \begin{pmatrix} a & b & b & c \\ b & b & a & b \\ a & a & b & b \\ c & a & b & b \end{pmatrix}$$

If we consider the strictly monotone functions $r, c : \{1, 2, 3\} \rightarrow \{1, 2, 3, 4\}$ such that $r(1) = c(1) = 1$, $r(2) = c(2) = 3$, $r(3) = c(3) = 4$, we have that $P_{ij} = Q_{r(i)c(j)}, \ \forall (i, j) \in \{1, 2, 3\} \times \{1, 2, 3, 4\}$, then $P \leq Q$.

It is clear to observe that $P \preceq Q$ if we can obtain P from Q by deleting some rows and/or columns.
In the previous example P is obtained from Q by deleting the second row and the second column.

This definition of subpicture generalizes to two-dimensional languages the notion of (scattered) *subword* of a word (cf.[13]). Indeed we can consider a word as a matrix of size $m \times 1$.

3.1 The Order on the Class of Discrete Sets

In case of binary alphabet we have an order relation between discrete sets and, in particular, polyominoes. With our notation we can, easily, say that if P and Q are two discrete sets such that P has dimension $m \times n$, then $P \leq Q$ if there are strictly monotone functions $r : \{1, ..., n\} \rightarrow \mathbb{N}_{\geq 1}$ and $c : \{1, ..., m\} \rightarrow \mathbb{N}_{\geq 1}$ such that $(i, j) \in P \Leftrightarrow (r(i), c(j)) \in Q$, for all $(i, j) \in \{1, ..., n\} \times \{1, ..., m\}$. See, for example, in Fig. 5 first polyomino is $\preceq$ than the second one. Indeed we can obtain first polyomino from the second one by deleting first and fifth columns and second row in any order.

Note 1. Note that the subpicture order $\preceq$ and the insiemistic inclusion order $\subseteq$ are two different binary relations on the class of discrete sets.

3.2 Down Sets

In this section we study the closure properties, with respect to the order, of the various families of polyominoes. The main theorem of the section (Theorem 2) gives a characterization of L-convex polyominoes in terms of the order $\preceq$: we prove that a polyomino is L-convex iff its down-set is contained in $\mathcal{P}$.

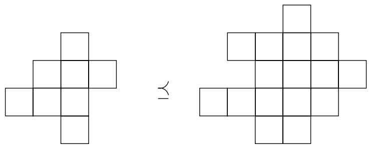


Fig. 5. Two comparable polyominoes.

Definition 3. *Let us consider the ordering* $(\mathcal{D}, \preceq)$ *and* $\mathcal{S}$ *a generic subset of* $\mathcal{D}$. $\mathcal{S}$ *is a down-set, with respect to* $\preceq$*, if it satisfies the following condition:*

$$if \;\; S \in \mathcal{S} \text{ and } T \preceq S, \text{ then } T \in \mathcal{S}.$$

We denote by $Down(\mathcal{S})$ the *down-set* of $\mathcal{S}$ in $(\mathcal{D}, \preceq)$:

$$Down(\mathcal{S}) = \{T \in \mathcal{D} |\; \exists\, S \in \mathcal{S} \text{ such that } T \preceq S\}.$$

If $S = \{P\}$ we denote by $Down(P)$ the down-set of S.

For any $\mathcal{S} \subseteq \mathcal{D}$ we have, in general, that $\mathcal{S} \subseteq Down(\mathcal{S})$ and we have that $\mathcal{S}$ is a down-set if $Down(\mathcal{S}) = \mathcal{S}$.

Our question is whether the families of polyominoes introduced in Section 2 are down-sets of $(\mathcal{D}, \preceq)$.

Proposition 2. $\mathcal{C}$ *is not a down set of* $(\mathcal{D}, \preceq)$.

Proof. The proof is given by the example in Fig. 6. Indeed, we have two discrete sets P and Q such that P is not a polyomino, Q is a convex polyomino and $P \preceq Q$. Then $Down(\mathcal{C}) \nsubseteq \mathcal{P}$.

This proves that $\mathcal{P}$ and $\mathcal{C}$ are not down-sets. We have following propositions.

Proposition 3. $Down(\mathcal{C}) \cap \mathcal{P} = \mathcal{C}$.

Proposition 4. $\mathcal{L}$ *is a down-set of* $(\mathcal{D}, \preceq)$*, i.e.* $Down(\mathcal{L}) = \mathcal{L}$

Proof. We have to prove that any element of $Down(\mathcal{L})$ is an L-convex polyomino. Let $P \in Down(\mathcal{L})$ then there exists an L-convex polyomino Q such that $P \preceq Q$. By definition, there are strictly monotone functions r and c such that for all (i, j), $(i, j) \in P \Leftrightarrow (r(i), c(j)) \in Q$. Let us observe that, since Q is convex its row and column are connected then rows and columns of P are connected too.

For any pair $P(i, j), P(h, k)$ of cell in P we have that $(r(i), c(j)) \in Q$ and $((r(h), c(k)) \in Q$. But Q is an L-convex polyomino then, by Lemma 1, we have that $((r(i), c(k)) \in Q$ or $((r(h), c(j)) \in Q$. We can conclude that $(i, k) \in P$ or $(j, h) \in P$ and, being ith and hth rows of P connected jth and kth columns of P connected, we have in P an L-path connecting $P(i, j)$ and $P(h, k)$. This proves that P is a polyomino and is L-convex.

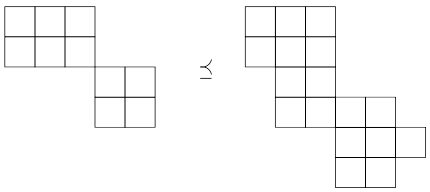


Fig. 6. A discrete set comparable with a convex polyomino.

Furthermore, we have a characterization of L-convex polyominoes as stated in following theorem.

Theorem 2. *Let $P \in \mathcal{D}$. P is L-convex if and only if $Down(P) \subseteq \mathcal{P}$.*

Proof. Let $P \in \mathcal{D}$. If P is L-convex polyominoes thesis follows from Proposition 4. Viceversa, since $Down(P) \subseteq \mathcal{P}$ P is a polyomino too and it is convex. We have to prove that P is L-convex. Let $(i,j),(h,k) \in P$, without loss of generality we can suppose that $h > i$ and $k > j$. Let Q such that $Q \preceq P$ with $r(i) = i$, $r(i+1) = h$, $c(j) = j$ and $c(j+1) = k$. Since $Q \in \mathcal{P}$ we have that $(i,j+1) \in Q$ or $(i+1,j) \in Q$ then $\forall (i,j),(h,k) \in P$, $(i,k) \in P$ or $(h,j) \in P$ proving P to be L-convex polyomino by Lemma 1.

4 Well-Ordering

We call *antichain* a set of pairwise incomparable elements. An ordering $(\mathcal{A}, \leq)$ is *well-ordering* if for every infinite sequence $A_1, A_2, A_3, ...$ from $\mathcal{A}$ there exist $i < j \in \mathbb{N}$ such that $A_i \leq A_j$.

There exist many equivalent conditions of well-ordering, in particular we have the following theorem (cf.[9]).

Theorem 3. *Let $(\mathcal{A}, \leq)$ be an ordering. $(\mathcal{A}, \leq)$ is a well-ordering iff every infinite sequence of elements of $\mathcal{A}$ has an infinite ascending subsequence.*

A basic theorem in the theory of well-quasi ordering is Higman's theorem (cf. [9]), which states, in particular, that subword ordering is a well-ordering in the set of words. Higman's theorem has been extended to structures more general than the words as, for instance, labelled trees (cf. [10]) and infinite words (cf. [7]). It is worth noting that there exist many papers devoted to applications of well ordering to formal language theory (cf., for instance,[6] and [5]).

Here we investigate whether there exists some extension of Higman's theorem to two dimensions. We consider the following hierarchy of partially ordered sets:

$$(\mathcal{R}, \preceq) \subseteq (\mathcal{L}, \preceq) \subseteq (\mathcal{C}, \preceq) \subseteq (\mathcal{P}, \preceq) \subseteq (\mathcal{D}, \preceq) \subseteq (\mathcal{M}, \preceq)$$

Recall that, if $(X, \leq)$ is a well-ordering and $Y \subseteq X$, then $(Y, \leq)$ is a well-ordering too.

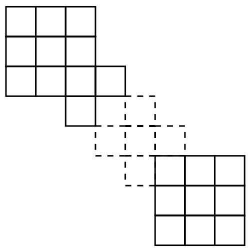


Fig. 7. Family of convex polyominoes.

Proposition 5. *$(\mathcal{C}, \preceq)$, $(\mathcal{P}, \preceq)$, $(\mathcal{D}, \preceq)$ and $(\mathcal{M}, \preceq)$ are not well-ordering.*

Proof. By previous remark, it suffices to prove that $(\mathcal{C}, \leq)$ is not a well-ordering.

Let $\{A_n\}_{n\geq 6}$ be the infinite family of symmetric square matrices A_n, of size $n \times n$, with entries $a_n(i,j)$, $i, j \leq n$ defined as follows

$$a_n(i,j) = \begin{cases} 1 & i-1 \leq j \leq i+1 \\ 1 & (i,j) \in \{(1,3),(3,1),(n-2,n),(n,n-2)\} \\ 0 & \text{otherwise} \end{cases}$$

Fig.7 shows convex polyominoes associated with matrices A_n. Let $n < m$ and $A_m \in \mathcal{A}$. To obtain the polyomino A_n from A_m we should delete $m-n$ rows and columns from A_m. It is easy to observe that by these operations we obtain a discrete set not belonging to our family. Then each element of $\{A_n\}_{n\geq 6}$ is incomparable to each other i.e. $\{A_n\}_{n\geq 6}$ is an infinite antichain.

Before proving (Theorem 4) that $(\mathcal{L}, \preceq)$ is a well-ordering we give some preliminaries.

A binary relation $\leq'$ on a set $\mathcal{A}$ is a *quasi-order* if it is reflexive and transitive. A quasi-order is *well-founded* if any strictly descending chain

$$A_0 \leq' A_1 \leq' \ldots \leq' A_n \leq' \ldots$$

of elements of $\mathcal{A}$, has a finite length.

Many proofs of well-ordering are based on the following proposition regarding existence of minimal of antichains in a well-found ordering sets (cf. [11]).

Proposition 6. *Let $\leq'$ be a well-founded quasi-order on $\mathcal{A}$. Let $\leq$ be a quasi-order on $\mathcal{A}$ which is not a well quasi-order. Then there exists an antichain (with respect to $\leq$) which is minimal respect to $\leq'$.*

Let Δ be the set of all antichains in $\mathcal{A}$ with respect to a quasi-order $\leq$ which is not a well quasi-order. Let $\leq'$ be a well-founded quasi-order on $\mathcal{A}$. An

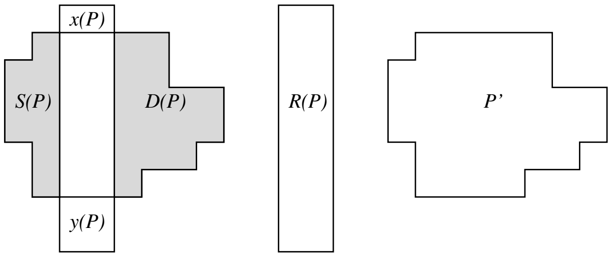


Fig. 8.

antichain $A_0, A_1, ..., A_n, ...$, with respect to $\leq$, is said to be *minimal*, with respect to $\leq'$, if A_0 is a minimal element, with respect to $\leq'$, of all first elements of the antichains in $\varDelta$ and, for all $i > 0$, A_{i+1} is a minimal element, with respect to $\leq'$, of all $(i+1)$-th elements of the antichains in $\varDelta$ having, as first i elements, $A_0, A_1, ..., A_i$.

Let us observe that to each L-convex polyomino P, with more than one maximal rectangle, we can associate a vector $(R(P), P', S(P), D(P), x(P), y(P))$ with $R(P) \in \mathcal{R}$, $P', L(P), R(P) \in \mathcal{L}$, and $x(P), y(P) \in \mathbb{N}$, defined as follows (see Fig. 8):

- $R(P)$ the element of $\mathcal{R}_{max}(P)$ with maximal height;
- $P' = P \setminus R(P)$ with the meaning stated in Section 2.2;
- $S(P)$ is the L-convex polyomino obtained from P by deleting R and columns after R;
- $D(P)$ is the L-convex polyomino obtained from P by deleting R and columns before R.
- $x(P)$ the number of rows of the top part of R not included in P';
- $y(P)$ the number of rows of the bottom part of R not included in P'.

Next lemma holds.

Lemma 2. *Let P_1 and P_2 be two L-convex polyominoes, $(R_1, P_1', S_1, D_1, x_1, y_1)$ and $(R_2, P_2', S_2, D_2, x_2, y_2)$ the vectors, respectively, associated. Then:*

$$\begin{cases} P_1' \preceq P_2' \\ R_1 \preceq R_2 \\ S_1 \preceq S_2 \\ D_1 \preceq D_2 \\ x_1 \leq x_2 \\ y_1 \leq y_2 \end{cases} \Rightarrow P_1 \preceq P_2$$

We are now ready to prove the main result of this section.

Theorem 4. *$(\mathcal{L}, \preceq)$ is a well-ordering.*

Proof. Let P be an L-convex polyomino, and $\rho(P) = |\mathcal{R}_{max}(P)|$, i.e. an integer value that counts the number of maximal rectangles of P. We define in $\mathcal{L}$ a quasi-order $\preceq'$ as follows:

$$P \preceq' Q \Leftrightarrow \rho(P) \leq \rho(Q).$$

This is a well-founded quasi-order.

Let us suppose, by contradiction, that $(\mathcal{L}, \preceq)$ is not a well ordering. By Proposition 6 there exists an antichain

$$s = P_0, P_1, P_2, \cdots, P_n, \cdots$$

minimal with respect to $\preceq'$. For any $i \in \mathbb{N}$, we can associate to each P_i the vector $(R_i, P'_i, S_i, D_i, x_i, y_i)$ so we can write:

$$P_i = P'_i \cup R(P_i) \text{ and } \quad 0 \leq \rho(P'_i) < \rho(P_i).$$

Where by $\cup$ we mean the overlapping. By reasoning on the lengths of rectangles, we can observe that $\mathcal{R}$ is a well ordering set, with respect to $\preceq$. We have, by Theorem 3, that the sequence $R(P_0), R(P_1), R(P_2), \cdots, R(P_n), \cdots$ has an infinite ascending subsequence

$$R(P_{i_1}) \preceq R(P_{i_2}) \preceq \cdots \preceq R(P_{i_n}) \cdots$$

with $1 \leq i_1 < i_2 < \cdots < i_n < \cdots$. The sequence

$$s' = P_0, P_1, \cdots, P_{i_1-1}, P'_{i_1}, P'_{i_2}, \cdots, P'_{i_n}, \cdots$$

is not an antichain, by minimality of s, then s' has an infinite ascending subsequence that we can suppose having first element with index greater or equal to i_1. Then it is

$$P'_{j_1} \preceq P'_{j_2} \preceq \cdots \preceq P'_{j_n} \cdots$$

with $i_1 \leq j_1 < j_2 < \cdots < j_n < \cdots$. Let us consider now the sequence

$$s_S = P_0, P_1, \cdots, P_{j_1-1}, S_{j_1}, S_{j_2}, \cdots, S_{j_n}, \cdots$$

where $S_{j_i} = S(P_{j_i})$ for any i. It is not antichain, by minimality of s, then, as before, there exists an infinite ascending subsequence

$$S_{k_1} \preceq S_{k_2} \preceq \cdots \preceq S_{k_n} \cdots$$

of s_S with $j_1 \leq k_1 < k_2 < \cdots < k_n \cdots$. In the same way, the infinite sequence

$$P_0, P_1, \cdots, P_{k_1-1}, D_{k_1}, D_{k_2}, \cdots, D_{k_n}, \cdots$$

where $D_{k_i} = D(P_{k_i})$ for any i, has an ascending subsequence

$$D_{h_1} \preceq D_{h_2} \preceq \cdots \preceq D_{h_n} \cdots$$

with $k_1 \leq h_1 < h_2 < \cdots < h_n \cdots$

Furthermore, since $\mathbb{N}$ is, trivially, a well-ordering the infinite sequence of integers

$$s_x = x_{h_1}, x_{h_2}, \cdots, x_{h_n}, \cdots$$

where $x_{h_i} = x(P_{h_i})$ for any i, has an infinite ascending subsequence

$$x_{f_1} \leq x_{f_2} \leq \cdots \leq x_{f_n} \cdots$$

with $h_1 \leq f_1 < f_2 < \cdots < f_n \cdots$. Finally, the corresponding infinite sequence of integers

$$y_{f_1}, y_{f_2}, \cdots, y_{f_n}, \cdots$$

with $y_{f_i} = y(P_{f_i})$ for any i, has an infinite ascending subsequence

$$y_{g_1} \leq y_{g_2} \leq \cdots \leq y_{g_n} \cdots$$

with $f_1 \leq g_1 < g_2 < \cdots < g_n \cdots$.

However we choose $i \leq j$, P_{g_i} and P_{g_j} are elements of s and, by Lemma 2, $P_{g_i} \preceq P_{g_j}$ which is a contradiction.

References

1. Barcucci, E., Del Lungo, A., Nivat, M., Pinzani, R.:Reconstructing convex polyominoes from horizontal and vertical projections. Theoret. Comput. Sci. **155** (1996) 321–347.
2. Bousquet-Mélou, M.: A method for the enumeration of various classes of column-convex polygons. Discrete Math. **155** (1996) 1–25.
3. Castiglione, G., Restivo, A.: Reconstruction of L-convex Polyominoes. Electronic Notes in Discrete Math.**12**, Elsevier Science (2003).
4. Castiglione, G., Frosini, A., Restivo, A., Rinaldi, S.: On L-convex Polyominoes, *submitted* to Theoret. Comp. Sci.(2004).
5. D'Alessandro, F., Varricchio, S.: Well quasi-orders on languages. Lecture Notes in Comp. Sci. **2710** (2003) 230–241.
6. Ehrenfencht, A., Haussler, D., Rozenberg, G.: On regularity of context-free languages. Theoret. Comput. Sci.**27** (1983) 311–332.
7. Finkel, A.: Une Généralisation des théorèmes de Higman et de Simon aux Mots Infinis. Theoret. Comput. Sci. **38** (1985) 137–142
8. Golomb, S.W.: Polyominoes. Scribner, New York (1965)
9. Higman, G.H.: Ordering by divisibility in abstract algebra. Proc. London Math. Soc. **2** (1952) 326–336
10. Kruskal, J.B.: Well-quasi-ordering, the Tree Theorem, and Vazsonyi's conjecture. Trans. Amer. Math. Soc. **95** (1960) 210–225
11. Kruskal, J.B.: The Theory of Well-Quasi-Ordering: A Frequently Discovered Concept. J. Combin. Theory Ser. A **13** (1972) 297–305
12. Kuba, A., Balogh, E.: Reconstruction of convex 2D discrete sets in polynomial time, Theoret. Comput. Sci. **283** (2002) 223–242.
13. Lothaire, M.: Combinatorics on Words. Encyclopedia of Mathematics and its Applications, **17**, Addison-Wesley, Reading, MA, (1983)
14. Matz, O.: On piecewise testable, starfree, and recognizable picture languages.Foundations of Software Science and Computation Structures **1378**(1998).

Subshifts Behavior of Cellular Automata. Topological Properties and Related Languages[⋆]

Gianpiero Cattaneo and Alberto Dennunzio

Università degli Studi di Milano–Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione,
Via Bicocca degli Arcimboldi 8, 20126 Milano (Italy)
{cattang, dennunzio}@disco.unimib.it

Abstract. We study the subshift behavior of one dimensional cellular automata and we show how to associate to any subshift of finite type a cellular automaton which contains it. The relationships between some topological properties of subshifts and the behavior of the related languages are investigated. In particular we focus our attention to the notion of full transitivity. We characterize the language related to a full transitive subshift extending the notion of irreducibility.

1 Introduction

A one–dimensional cellular automaton (CA) is a bi–infinite array of identical elements, called cells, located on a straight line and whose position or site is labelled by an integer number $i \in \mathbb{Z}$. Each cell can assume a value chosen from a finite set $\mathcal{A}$, the alphabet of the CA, and changes its value according to a local rule f, homogeneously applied to all cells of the automaton, in a discrete time evolution. Let us recall that a Discrete Time Dynamical System (DTDS) is a pair $\langle X, g\rangle$, where the state space X is a nonempty set equipped with a distance d and the next state mapping $g : X \mapsto X$ is continuous on X with respect to the metric d. A CA can be viewed as a DTDS $\left\langle \mathcal{A}^{\mathbb{Z}}, F_f \right\rangle$ whose state space is the set $\mathcal{A}^{\mathbb{Z}}$, also called the set of the configurations on alphabet $\mathcal{A}$. The next state mapping F_f of a CA is the global transition mapping induced by local rule f.

The empirical observation of one dimensional CA leads to realize that they share a subshift behavior. This means that on a subset S_f of CA configurations the global transition mapping F_f coincides with the left shift mapping σ. The DTDS $\langle S_f, \sigma\rangle$ is called the subshift contained in the CA whose local rule is f.

Symbolic dynamics, such as subshifts, has found significant application in different disciplines, e.g., data storage and transmission, coding and linear algebra [5]. Subshifts are also studied in language theory. Indeed to any subshift it is associated a formal language. An important class of subshift is constituted by the subshifts of finite type (SFT), i.e., subshifts which can be described by a finite set of words and represented by a directed graph.

⋆ This work has been supported by M.I.U.R. COFIN project "Formal Languages and Automata: Theory and Application"

Subshifts contained in CA are SFT. In this work, we will show that, given an SFT, it is possible to construct one or more suitable CA which contain it. We also study some topological properties of DTDS and apply this investigation in the case of SFT, focusing our attention to the properties of corresponding language, graph and matrix. We are interested to the several definitions of "transitivity" which can be found in literature. In particular, we rename the Devaney notion given in [4] as positive transitivity and consider the notion of full transitivity, which, in the case of compact and homeomorphic DTDS, can be found in [3, 6] with other names.

This paper is organized as follows. In section 2 we give basic definitions and notions. In section 3 we consider the subshifts contained in CA. In section 4 the study of subshifts made in [2] is extended to full transitivity. The language, the matrix, and the graph associated to a full transitive subshift are characterized. We also study the relationships between sensitivity to the initial condition of subshifts and associated languages.

2 Basic Definitions

2.1 Cellular Automata, Subshifts and Languages

Definition 1 (One Dimensional CA). *A one dimensional bi-infinite CA is a triple $\langle \mathcal{A}, r, f \rangle$, where $\mathcal{A}$ is the alphabet, $r \in \mathbb{N}$ is the* radius*, and $f : \mathcal{A}^{2r+1} \mapsto \mathcal{A}$ is the* local rule*, on the basis of which each cell is updated.*

The simplest case is the boolean one of *elementary cellular automata* (ECA) characterized by $r = 1$. The different $2^{2^3} = 256$ ECA are classified by the natural number $n_f = f(0,0,0) \cdot 2^0 + f(0,0,1) \cdot 2^1 + \cdots + f(1,1,1) \cdot 2^7$.

The set $\mathcal{A}^{\mathbb{Z}}$ of all configurations of a CA can be equipped with the Tychonoff metric, $d_T(\underline{x}, \underline{y}) = \sum_{i=-\infty}^{+\infty} \frac{1}{4^{|i|}} \delta(x_i, y_i)$ where δ is the Hamming distance on $\mathcal{A}$. So a CA induces a compact, perfect, and complete DTDS $\left\langle \mathcal{A}^{\mathbb{Z}}, F_f \right\rangle$ where the global transition mapping $F_f : \mathcal{A}^{\mathbb{Z}} \mapsto \mathcal{A}^{\mathbb{Z}}$ associates with any configuration $\underline{x} \in \mathcal{A}^{\mathbb{Z}}$ the next time step configuration $F_f(\underline{x})$ whose i-th component is expressed by the local rule according to: $[F_f(\underline{x})]_i = f(x_{i-r}, \ldots, x_i, \ldots, x_{i+r})$. Let us remark that the global mapping F_{170} induced by the ECA local rule 170 coincides with the left shift mapping $\sigma : \mathcal{A}^{\mathbb{Z}} \mapsto \mathcal{A}^{\mathbb{Z}}$ $(\forall \underline{x} \in \mathcal{A}^{\mathbb{Z}}, \forall i \in \mathbb{Z}, [\sigma(\underline{x})]_i = x_{i+1})$.

We define the cylinder of block $u \in \mathcal{A}^n$ and position $m \in \mathbb{Z}$ as the set $C_m(u) = \{\underline{x} \in \mathcal{A}^{\mathbb{Z}} \mid x_m \cdots x_{m+n-1} = u\}$. Note that cylinders form a basis of clopen subsets of $\mathcal{A}^{\mathbb{Z}}$ for the topology induced by the Tychonoff metric.

Definition 2 (Subshift). *A (2-sided)* subshift *over the alphabet $\mathcal{A}$ is a DTDS $\langle S, \sigma_S \rangle$, where S is a non empty closed, strictly σ-invariant ($\sigma(S) = S$) subset of $\mathcal{A}^{\mathbb{Z}}$, and σ_S is the restriction of the shift map σ to S.*

In the sequel, in the context of a given subshift $\langle S, \sigma_S \rangle$, for the sake of simplicity we will denote by $C_m(u)$ the subset $C_m(u) \cap S$, if there is no confusion. Let us note that in relative topological space (S, d_T), where d_T is the restriction to S of the Tychonoff metric defined on $\mathcal{A}^{\mathbb{Z}}$, the set $C_m(u) \cap S$ is open.

A subshift $\langle S, \sigma_S \rangle$ distinguishes the words or finite blocks constructed over the alphabet $\mathcal{A}$ in two types: *admissible blocks*, i.e., blocks appearing in some configuration of S and blocks which are not admissible, called *forbidden*, i.e., blocks which do not appear in any configuration of S. We will write $w \prec \underline{x}$ to denote that the $\mathcal{A}$–word $w = (w_1, \ldots, w_n) \in \mathcal{A}^*$ appears in the configuration $\underline{x} \in \mathcal{A}^{\mathbb{Z}}$, formally $\exists i \in \mathbb{Z}$ s.t. $x_i = w_1, \ldots, x_{i+n-1} = w_n$ (also indicated by $\underline{x}_{[i,i+n-1]} = w$). We will denote by $w \not\prec \underline{x}$ the fact that w does not appear in the configuration $\underline{x}$. A word $u = u_1 \cdots u_m \in \mathcal{A}^*$ is a sub-block (or sub-word) of $w = w_1 \cdots w_n \in \mathcal{A}^*$, written as $u \sqsubseteq w$, iff $u = w_i \cdots w_j$, for some $1 \leq i \leq j \leq n$. To every subshift we can associate a formal language according to the following:

Definition 3 (Language of a Subshift). *Let $\langle S, \sigma_S \rangle$ be a subshift over the alphabet $\mathcal{A}$. The* language *of S is the collection of all admissible blocks: $\mathcal{L}(S) = \{w \in \mathcal{A}^* : \exists \underline{x} \in S \text{ s.t. } w \prec \underline{x}\}$.*

A canonical way to generate a subshift consists of fixing a collection of words considered as forbidden blocks. Precisely, let $\mathcal{F}$ be any subset of $\mathcal{A}^*$, and let us construct the set $S(\mathcal{F}) = \{\underline{x} \in \mathcal{A}^{\mathbb{Z}} : \forall w \in \mathcal{F}, w \not\prec \underline{x}\}$. Then $S(\mathcal{F})$ is a subshift, named the subshift generated by $\mathcal{F}$. Let us note that two different families of forbidden blocks may generate the same subshift.

Definition 4 (Subshift of Finite Type). *A subshift is of finite type iff it can be generated by a finite set $\mathcal{F}$ of forbidden blocks.*

In the case of a SFT, the finite set $\mathcal{F}$ generally is composed by blocks of different length. Nevertheless, starting from $\mathcal{F}$ it is always possible to construct a set of forbidden blocks $\mathcal{F}'$ consisting of the same length and generating the same subshift. We have just to complete in all possible ways each block from $\mathcal{F}$, up to reach the length of the longest forbidden block in $\mathcal{F}$. In the case of a SFT $\langle S, \sigma_s \rangle$ we will denote by $\mathcal{F}_h$ a set of forbidden blocks in which all words $w \in \mathcal{F}_h$ have the same length h and generating the subshift, i.e. $S = S(\mathcal{F}_h)$.

To every SFT we can associate a graph according to the following:

Definition 5 (Graph associated to a SFT). *Let $\langle S, \sigma_S \rangle$ be a SFT generated by a set $\mathcal{F}_h$ of forbidden blocks. The* graph *$G_h(S)$ associated to S is the pair $\langle V(S), E(S) \rangle$ where the vertex set is $V(S) = \mathcal{A}^{h-1}$ and the edge set $E(S)$ contains all the pairs $(a, b) \in \mathcal{A}^{h-1} \times \mathcal{A}^{h-1}$ such that $a_2 = b_1, \ldots, a_{h-1} = b_{h-2}$ and $a_1 a_2 \cdots a_{h-1} b_{h-1} \notin \mathcal{F}_h$. The block $a_1 a_2 \cdots a_{h-1} b_{h-1}$, denoted also by $a \odot b$, is called the word generated by the blocks a and b.*

Trivially, bi-infinite paths along nodes on the graph $G_h(S)$ correspond to bi-infinite strings of the SFT S. In the general case we can minimize the subgraph $G_h(S)$ removing all the unnecessary nodes and the corresponding outgoing and incoming edges. In this way the finite paths along nodes on the graph correspond to the words of the language $\mathcal{L}(S)$. From now on, we will consider only minimized graphs. We will denote by $A_h(S)$ the adjacency matrix of the graph $G_h(S)$ associated to a SFT $\langle S, \sigma_S \rangle$ generated by a set $\mathcal{F}_h$ of forbidden blocks.

We recall now the following definitions concerning the notion of irreducibility for a language and a square matrix.

Definition 6 (Irreducible Language). *A language $\mathcal{L} \subseteq \mathcal{A}^*$ is* irreducible *iff for every ordered pair of blocks $u, v \in \mathcal{L}$ there is a block $w \in \mathcal{L}$ such that $uwv \in \mathcal{L}$.*

Definition 7 (Irreducible Matrix). *An order k square matrix $M = [m_{i,j}]$ is irreducible iff $\forall i, j \in \{1, \ldots, k\}$, $\exists p = p(i,j) \in \mathbb{N}, p > 0$ such that $m_{i,j}^{(p)} \neq 0$, where $m_{i,j}^{(p)}$ is the (i,j)-component of the matrix M^p.*

2.2 Topological Properties of Discrete Time Dynamical Systems

In this section we give the definitions of some topological properties which describe some behaviors of general DTDS.

Definition 8 (Positive Transitivity). *A DTDS $\langle X, g \rangle$ is (topologically)* positively transitive *iff for any pair A and B of non empty open subsets of X there exists a natural number $n > 0$ such that $g^n(A) \cap B \neq \emptyset$.*

Intuitively, a positively transitive map g has points which eventually move under iteration of g from one arbitrarily small neighborhood to any other. As a consequence, the dynamical system cannot be decomposed into two disjoint clopen sets which are invariant under the iterations of g. On compact DTDS $\langle X, g \rangle$ with $g(X) = X$, positive transitivity is equivalent to the existence of a dense orbit and in [6, p. 127] is called *one-sided* topological transitivity.

We now recall another notion of transitivity which was introduced for homeomorphic DTDS on compact spaces (see for instance [3, p. 31] where it is simply called transitivity) and which can be applied to a general DTDS. We rename this kind of transitivity in order to avoid confusion with the positive one defined above, also if in this paper it will be used only in the case of homeomorphic DTDS on compact spaces:

Definition 9 (Full Transitivity). *A DTDS $\langle X, g \rangle$ is (topologically)* full transitive *iff for any pair A and B of non empty open subsets of X there exists an integer number $t \in \mathbb{Z}$ such that $g^t(A) \cap B \neq \emptyset$ (which is equivalent to $A \cap g^{-t}(B) \neq \emptyset$)*

It is obvious that homeomorphic DTDS on compact spaces which are positively transitive are full transitive too. The two following definitions refer to stronger conditions than positive transitivity.

Definition 10 (Topological Mixing). *A DTDS $\langle X, g \rangle$ is* topologically mixing *iff for any pair A and B of non empty open subsets of X there exists a natural number n_0 such that for every $n \geq n_0$ we have $g^n(A) \cap B \neq \emptyset$.*

Definition 11 (Strong Transitivity). *A DTDS $\langle X, g \rangle$ is* strongly transitive *iff for all nonempty open set $A \subseteq X$ we have $\bigcup_{n=0}^{+\infty} g^n(A) = X$.*

A strongly transitive map g has points which eventually move under iteration of g from one arbitrarily small neighborhood to any other *point*. We recall now a property which is often referred to as an element of regularity of DTDS.

Definition 12 (Regularity). *A DTDS $\langle X, g\rangle$ is* regular *iff the set of its periodic point $Per(g) = \{p \in X \mid \exists n \in \mathbb{N}, n > 0 : g^n(p) = p\}$ is dense in X.*

The following notion is recognized as a central notion in chaos theory since it captures the feature that in chaotic systems small errors in experimental readings might lead to large scale divergence.

Definition 13 (Sensitivity to the Initial Conditions). *A DTDS $\langle X, g\rangle$ is* sensitive to the initial conditions *iff there exists a constant $\epsilon > 0$ such that for any state $x \in X$ and for any $\delta > 0$ there exists a state $y \in X$ and an integer $t_0 \in \mathbb{N}$ such that $d(x, y) < \delta$ and $d(g^{t_0}(x), g^{t_0}(y)) \geq \epsilon$. Constant ϵ is called the sensitivity constant of the system.*

The popular book by Devaney [4] isolates three components as being the essential features of chaos: positive transitivity, regularity and sensitivity to the initial conditions . In [1] it has been shown that if a DTDS with infinite cardinality is regular and positively transitive, then it is also sensitive to the initial conditions.

3 Subshifts Contained in CA

Now we illustrate a sufficient condition on the local rule of a CA in order that the global dynamic turns out to be a subshift. For this goal, a suitable set of forbidden blocks with respect to the local CA rule will be constructed.

Definition 14. *Let $\langle \mathcal{A}, r, f\rangle$ be a CA. A block $w = w_{-r} \dots w_0\, w_1 \dots w_r \in \mathcal{A}^{2r+1}$ is called left-forbidden with respect to the CA local rule f iff $f(w) \neq w_1$.*

Proposition 1. [2] *Let $\langle \mathcal{A}, r, f\rangle$ be a CA and $\left\langle \mathcal{A}^{\mathbb{Z}}, F_f\right\rangle$ be the associated DTDS. Let $\mathcal{F}_{2r+1}(f) = \{w_{-r} \dots w_0 \dots w_r \in \mathcal{A}^{2r+1} : f(w_{-r} \dots w_0 \dots w_r) \neq w_1\}$ be the set of all left-forbidden blocks of the local rule f and let $S_f = \{\underline{x} \in \mathcal{A}^{\mathbb{Z}} : \forall w \in \mathcal{F}_{2r+1}(f), w \not\prec \underline{x}\}$ be the set of all configurations which do not contain any local rule left-forbidden block. Then $\langle S_f, F_f\rangle$ is a subshift, called the subshift contained in the CA.*

In order to represent the CA subshifts, we consider the graph $G_{2r+1}(S_f) = \langle V(S_f), E(S_f)\rangle$ and we label it by letters of the alphabet $\mathcal{A}$ using the local rule f of the involved CA. Precisely, the label of each edge $(x, y) \in E(S)$ is defined as $l(x, y) := f(x_1, x_2, \dots, x_{2r}, y_{2r}) \in \mathcal{A}$ (where $(x_1, x_2, \dots, x_{2r}, y_{2r}) \notin \mathcal{F}_{2r+1}(f)$). This labelled graph will be denoted by $\mathcal{AG}_f$. Trivially, bi-infinite paths along nodes on the graph $\mathcal{AG}_f$ correspond to bi-infinite strings of the subshift S_f.

Proposition 2. *For any SFT $\langle S, \sigma_S\rangle$ there exists at least a CA $\langle \mathcal{A}, r, f\rangle$ which contains it.*

Proof. Let $\mathcal{F}_h$ be a set of forbidden block generating the SFT $\langle S, \sigma_S\rangle$. If h is odd, let r be the positive integer such that $h = 2r + 1$. Let us construct the local rule f as follows: for any $w = w_{-r} \dots w_0 \dots w_r \in \mathcal{F}_h$, $f(w) = b$, with $b \neq w_1$, and for any $w \in \mathcal{A}^h \setminus \mathcal{F}_h$, $f(w) = b$, with $b = w_1$. We have that $\mathcal{F}_{2r+1}(f) = \mathcal{F}_h$ and then $S = S_f$. If h is even, it is possible to construct a set of forbidden blocks of odd length $h+1$ generating the same SFT, and one can proceed as described above.

We want to stress that there could be different CA containing the same SFT. Indeed, for any forbidden block w, we could set as value of $f(w)$ just w_1 (see the proof) if this choice did not generate bi-infinite paths on the graph $\mathcal{AG}_f$ which are not also on the graph $G_h(S)$. Moreover, in the particular case $|\mathcal{A}| > 2$, for each forbidden block w there are $|\mathcal{A}| - 2$ different choices to set the value of $f(w)$. As consequence of the previous results we can state that the class of SFT coincides with the class of SFT contained in CA.

4 Properties of Subshift and Related Languages

In this section we want to study some topological properties of subshifts as DTDS, characterizing the corresponding languages, and for SFT, the associated matrixes and graphs too. We start focusing our attention to the notions of positive and full transitivity. Before studying these properties in the case of subshifts, we note that they are equivalent notions for homeomorphic CA. This result is due to the fact that homeomorphic CA are regular, and then each configuration $\underline{x} \in \mathcal{A}^{\mathbb{Z}}$ is a non-wandering point for the global mapping F_f (i.e., for any neighborhood U of $\underline{x}$ there exists an integer $n > 0$ s.t. $F_f^{-n}(U) \cap U \neq \emptyset$, see [6]). However we will see that (trivially outside homeomorphic CA) there exist full transitive subshifts which are not positively transitive.

In [2] three different techniques are explained to investigate if a SFT is positively transitive. These techniques involve the language, the graph, and the matrix associated to the SFT.

Theorem 1. **[2]** *Let $\langle S, \sigma_S \rangle$ be a SFT generated by a set of forbidden blocks $\mathcal{F}_h$. Then the following statements are equivalent: i) $\langle S, \sigma_S \rangle$ is positively transitive; ii) $\mathcal{L}(S)$ is irreducible; iii) $G_h(S)$ is strongly connected; iv) $A_h(S)$ is irreducible. Moreover, if one of the above equivalent conditions holds, then $\langle S, \sigma_S \rangle$ is regular. Lastly, if $\langle S, \sigma_S \rangle$ has infinite cardinality, the previous statements are equivalent to the following condition: v) $\langle S, \sigma_S \rangle$ is chaotic according to Devaney.*

Let us stress that the equivalence between *i)* and *ii)* holds even if the subshift is not a SFT. In [2] it has been also proved that the class of transitive SFT contained in ECA is the union of the class of topologically mixing ECA-subshifts and the class of strongly transitive ECA-subshifts. We now introduce some new definitions concerning languages, graphs, and matrices which help us to establish if a subshift is full transitive.

Definition 15 (Weakly irreducible language).
A language $\mathcal{L} \subseteq \mathcal{A}^*$ is weakly irreducible *iff for any pair of blocks $u, v \in \mathcal{L}$ there is a block $w \in \mathcal{L}$, s.t. $u, v \sqsubseteq w$.*

Example 1. A weakly irreducible language which is not irreducible.
Let $\langle S, \sigma_S \rangle$ be the SFT over the alphabet $\mathcal{A} = \{0, 1\}$ generated by the set of forbidden blocks $\mathcal{F}_2 = \{10\}$. The language $\mathcal{L}(S)$ is weakly irreducible. However

$$\begin{tikzcd} 0 \arrow[loop left] \arrow[r] & 01 \arrow[r] & 1 \arrow[loop right] \end{tikzcd} \qquad \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

Fig. 1. Graph and adjacency matrix associated to the SFT of example 1

it is not irreducible. Indeed, if we consider the words $u = 1 \in \mathcal{L}(S)$ and $v = 0 \in \mathcal{L}(S)$ we are not able to find any block $w \in \mathcal{L}(S)$ such that $uwv \in \mathcal{L}(S)$.

In the sequel we will refer to a connected component of a directed graph G as a subgraph which is a connected component of the underlying undirected graph of G obtained suppressing the orientations of all the edges of G. We now introduce the following:

Definition 16 (Full Transitive Graph). *A graph G is* full transitive *iff either it is strongly connected or it is* bicyclic, *i.e., a graph of the kind $\langle V, E \rangle$ where the vertex set is $V = \{U_0, \ldots, U_{m-1} = T_0, \ldots, T_i, \ldots, T_l = W_0, \ldots, W_{n-1}\}$ $(m, n, l \in \mathbb{N} \setminus \{0\})$ and the edges are the pairs $(U_i, U_{(i+1) \bmod m})$, for $i = 0, \ldots, m-1$, the pairs (T_i, T_{i+1}), for $i = 0, \ldots, l-1$ and the pairs $(W_i, W_{(i+1) \bmod n})$, for $i = 0, \ldots, n-1$.*

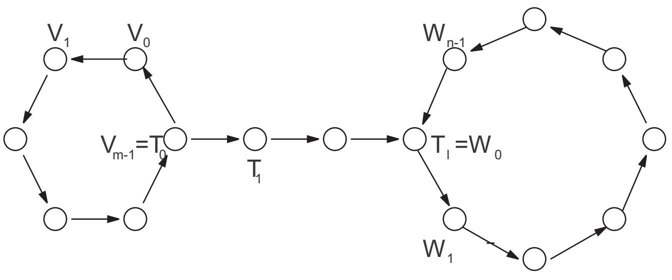


Fig. 2. Bicyclic graph: a full transitive graph which is not strongly connected

In other words, a full transitive graph is a unique connected component which is either strongly connected or it is constituted by two disjoint cycles and by a unique path which connects them.

Definition 17 (Full Transitive Matrix). *A square matrix $M = [m_{i,j}]$ is* full transitive *iff either it is irreducible or it is* bicyclic, *i.e., a matrix of the kind*

$$M = \begin{bmatrix} l_{1,1} & \cdots & l_{1,m} & 0 & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ l_{m,1} & \cdots & l_{m,m} & 1 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 1 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 1 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & \cdots & r_{1,1} & \cdots & r_{1,n} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & r_{n,1} & \cdots & r_{n,n} \end{bmatrix} \qquad (1)$$

where $L = [l_{i,j}]$ and $R = [r_{i,j}]$ are permutation matrices of order m and n respectively, which have the element 1 in the positions $(m, 1), (i, i + 1)$ for $i = 1, \ldots, m - 1$ and $(n, 1), (j, j + 1)$ for $j = 1, \ldots, n - 1$ respectively, or there exist a permutation matrix P such that $P^{-1}MP$ has the the structure expressed in 1.

Theorem 2. *Let $\langle S, \sigma_S \rangle$ be a subshift. Then the following statements are equivalent: i) $\langle S, \sigma_S \rangle$ is full transitive; ii) $\mathcal{L}(S)$ is weakly irreducible. In the case of a SFT generated by a set $\mathcal{F}_h$ of forbidden blocks, the previous statements are equivalent to the further conditions: iii) $G_h(S)$ is full transitive; iv) $A_h(S)$ is full transitive.*

Proof. i) ⇒ ii) Chosen arbitrarily $u, v \in \mathcal{L}(S)$, there exist a configuration $\underline{z} \in C_0(u)$ and an integer $t \in \mathbb{Z}$ such that $\sigma_S^t(\underline{z}) \in C_n(v)$, where $n = |u|$. Since $\sigma_S^t(\underline{z}) \in C_{-t}(u)$, we have that the words u and v are sub-blocks of the word $w = \sigma_S^t(\underline{z})_{[\min\{-t,n\},\max\{-t+n-1,n+|v|-1\}]} \in \mathcal{L}(S)$.
ii) ⇒ i) Chosen arbitrarily $u, v \in \mathcal{L}(S)$ and $m, n \in \mathbb{Z}$, let $w \in \mathcal{L}(S)$ be the block such that $u, v \sqsubseteq w$ with $u = w_i \cdots w_j$ and $v = w_k \cdots w_l$, for some $1 \leq i \leq j \leq |w|$ and some $1 \leq k \leq l \leq |w|$. Let us consider a configuration $\underline{z} \in C_{m-i+1}(w)$, we have that $\underline{z} \in C_m(u)$ and then $\sigma_S^t(\underline{z}) \in C_m(v)$, with $t = m - n + k - i \in \mathbb{Z}$.
ii) ⇒ iii) Let us assume that *ii)* is true in a SFT whose language is not irreducible. Condition that $\mathcal{L}(S)$ is weakly irreducible implies for every pair of strongly connected component (SCC for short) of $G_h(S)$ there must exist a path which connects them, thus $G_h(S)$ is constituted by a unique connected component which is not strongly connected. We now prove that for every pair $\mathcal{S}_1, \mathcal{S}_2$ of distinct SCC of $G_h(S)$ there exist a unique path connecting $\mathcal{S}_1$ to $\mathcal{S}_2$. Let $\pi_1 = V_1, \ldots V_m$ ($V_1 \in \mathcal{S}_1, V_2 \in \mathcal{S}_1$) be a simple path connecting $\mathcal{S}_1$ to $\mathcal{S}_2$, written as $\mathcal{S}_1 \longrightarrow^{\pi_1} \mathcal{S}_2$. For the sequel of argument, let $\pi_2 \neq \pi_1$, with $\mathcal{S}_1 \longrightarrow^{\pi_2} \mathcal{S}_2$, be a simple path having V_1 and V_m as first and last vertex, respectively. If u and v are the words generated by the vertexes of π_1 and π_2 respectively, then there exists a block $w \in \mathcal{L}(S)$ such that $u, v \sqsubseteq w$. This fact means that either $\mathcal{S}_1$ and $\mathcal{S}_2$ are subgraphs of the same SCC or π_2 is not a simple path or $\pi_2 = \pi_1$. In all these cases we obtain a contradiction. We now prove that every SCC of $G_h(S)$

is a cycle. Let us assume that $\mathcal{S}_1$ is a non cyclic SCC and let V_1 be a vertex in $\mathcal{S}_1$ having (at least) two incoming distinct edges (L_1, V_1) and (L_2, V_1). Let $T_0, T_1, \ldots, T_l$ ($T_i \notin \mathcal{S}_1$ and $T_i \notin \mathcal{S}_2$ for $0 < i < l$) be the path connecting $\mathcal{S}_1$ to an other SCC $\mathcal{S}_2$ and let $\pi = (V_1, \ldots, T_0)$ be a path of least length from V_1 to T_0. Considering the two paths $\pi_1 = L_1, \pi T_1$ and $\pi_2 = L_2, \pi, T_1$, if u and v are the words generated by the vertexes of π_1 and π_2 respectively, since there is a block w such that $u, v \sqsubseteq w$, either $\mathcal{S}_1$ and $\mathcal{S}_2$ are subgraph of the same SCC or $L_1 = L_2$ or $T_1 = T_0$ or $T_1 \in \mathcal{S}_1$. In all these cases we obtain a contradiction. Finally we prove that $G_h(S)$ is constituted only by two cyclic SCC $\mathcal{S}_1$ and $\mathcal{S}_2$. Let us assume that there exists a cyclic SCC $\mathcal{S}_3$. We treat the following case: $\mathcal{S}_1 \longrightarrow^{\pi_1} \mathcal{S}_2 \longrightarrow^{\pi_2} \mathcal{S}_3$. Let us consider two arbitrary vertexes $V_1 \in \mathcal{S}_1$, $V_3 \in \mathcal{S}_3$. Let $\pi = V_1, \ldots, W_1, \ldots, W_m, \ldots, V_3$ be the simple path where $W_1, \ldots, W_m \in \mathcal{S}_2$. Let $\underline{x} = (a^*)$, with $a \in \mathcal{L}(S)$, be the configuration obtained as a bi-infinite path on the vertexes of the cycle $\mathcal{S}_2$. Let $u = a \ldots a \in \mathcal{L}(S)$ be the word obtained repeating the block a in a such way that $|u| > |c|$ where $c \in \mathcal{L}(S)$ is the block generated by the vertexes $W_1, \ldots, W_m$. If v is the block generated by the vertexes of π, then it is impossible to find a word $w \in \mathcal{L}(S)$ such that $u, v \sqsubseteq w$. The other cases can be treated in a similar way.
iii) $\Rightarrow$ *i)* Let us consider a SFT whose graph $G_h(S)$ is not strongly connected. Then there is a configuration $\underline{z} = (a^* b c^*) \in S$ obtained as a bi-infinite path on $G_h(S)$, for suitable $a, b, c \in \mathcal{L}(S)$. We have that any word $w \in \mathcal{L}(S)$ is such that $w \prec \underline{z}$ and any configuration $\underline{x} \in S$ is of the kind $\sigma_S^t(\underline{z})$ for some $t \in \mathbb{Z}$. Let us choose two arbitrary blocks $u, v \in \mathcal{L}(S)$ and two arbitrary integers $m, n \in \mathbb{Z}$. Then there exist two configurations $\underline{x} = \sigma^{t_1}(\underline{z}) \in C_m(u)$ and $\underline{y} = \sigma_S^{t_2}(\underline{z}) \in C_n(u)$, for some $t_1, t_2 \in \mathbb{Z}$. Thus we obtain $\sigma_S^{t_2 - t_1}(\underline{x}) = \underline{y} \in C_n(u)$.
iii) $\Leftrightarrow$ *iv)* It directly follows from the structure of $G_h(S)$ and $A_h(S)$.

Example 2. A full transitive subshift which is not positively transitive.
Let us consider the subshift of the example 1. Since the graph is not strongly connected the subshift is not positively transitive. However it is full transitive.

It is easy to prove that a SFT generated by a set $\mathcal{F}_h$ of forbidden blocks is regular iff every connected component of the graph $G_h(S)$ is a SCC (see [5, p. 213]). This fact implies that a full but non positively transitive SFT is not regular. We now illustrate some conditions under which a subshift is sensitive to the initial conditions. For this goal, let us introduce a suitable notion of language.

Definition 18 (Right (resp., Left) 2-ways Extendible Language). *A language $\mathcal{L} \subseteq \mathcal{A}^*$ is* right (resp., left) 2-ways extendible *iff for any block $u \in \mathcal{L}$ there exist two words $w, w' \in \mathcal{L}$, with $w \neq w'$ and $|w| = |w'|$, such that $uw, uw' \in \mathcal{L}$ (resp., $wu, w'u \in \mathcal{L}$).*

Definition 19 (Right (resp., Left) 2-ways Extendible Path). *A path $\pi = V_1, \ldots, V_m$ $(m > 1)$ on a graph G is* right (resp., left) 2-ways extendible *iff there exists two paths $\pi' = V_1, \ldots, V_m, R'_1, \ldots, R'_n$ and $\pi'' = V_1, \ldots, V_m, R''_1, \ldots, R''_n$*

$(n \geq 1)$ (resp., $\pi'_1 = L'_1, \ldots, L'_n, V_1, \ldots, V_m$ and $\pi''_1 = L''_1, \ldots, L''_n, V_1, \ldots, V_m$) such that $R'_i \neq R''_i$ for some i (resp., $L'_i \neq L''_i$ for some i).

We recall that a sensitive DTDS must be perfect (that is, without isolated points), and as regards to perfectness of subshift, we give the following result whose proof is similar to the one of Theorem 3:

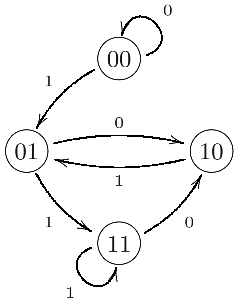
Proposition 3. *A subshift is perfect iff its language is either left or right 2-ways extendible. Moreover a SFT generated by a set $\mathcal{F}_h$ of forbidden blocks is perfect iff every path on the graph $G_h(S)$ is either left or right 2-ways extendible.*

Theorem 3. *Let $\langle S, \sigma_S \rangle$ be a subshift. Then the following statements are equivalent: $i)$ $\langle S, \sigma_S \rangle$ is sensitive to the initial conditions with sensitivity constant $\epsilon = 1$; $ii)$ $\mathcal{L}(S)$ is right 2-ways extendible. Moreover, in the case of a SFT generated by a set $\mathcal{F}_h$ of forbidden blocks, the previous statements are equivalent to the following condition: $iii)$ every path on the graph $G_h(S)$ right 2-ways extendible.*

Proof. $i) \Rightarrow ii)$ Chosen an arbitrary block $u \in \mathcal{L}(S)$ with $|u| = n$, let us consider a configuration $\underline{x} \in C_1(u)$. Then there must exist a configuration $\underline{y} \in C_{-n}(\underline{x}_{[-n,n]})$ and an integer $t \in \mathbb{N}$ such that $d_T(\sigma_S^t(\underline{x}), \sigma_S^t(\underline{y})) \geq 1$. This fact implies that $x_t \neq y_t$ (necessarily we have $t > n$). Introducing the distinct words $w' = \underline{x}_{[n+1,t]}$ and $w'' = \underline{y}_{[n+1,t]}$, we obtain that $uw, uw' \in \mathcal{L}(S)$.
$ii) \Rightarrow i)$ For any configuration $\underline{x} \in S$ and any integer $n \in \mathbb{N}$, there exist two distinct blocks $w' \in \mathcal{L}(S)$ and $w'' \in \mathcal{L}(S)$ such that $x_{[-n,n]}w', x_{[-n,n]}w'' \in \mathcal{L}(S)$. Thus there is a configuration $\underline{y} \in C_{-n}(x_{[-n,n]})$, with $y_i \neq x_i$ for some $i > n$.
$ii) \Rightarrow iii)$ Let us consider a SFT. Let $\pi = V_1, \ldots, V_m$ be a path on the graph $G_h(S)$ and let $u \in \mathcal{L}(S)$ be the block generated by the words corresponding to the vertexes of π. Then there exist two distinct words $w', w'' \in \mathcal{L}(S)$, with $|w'| = |w''|$, such that $uw', uw'' \in \mathcal{L}(S)$. This fact implies that the two paths $\pi' = V_1, \ldots, V_m, R'_1, \ldots, R'_n$ and $\pi'' = V_1, \ldots, V_m, R''_1, \ldots, R''_n$ generating the words uw' and uw'' respectively, are such that $R'_i \neq R''_i$ for some i.
$iii) \Rightarrow i)$ Chosen an arbitrary configuration $\underline{x} \in S$ and an integer $m \in \mathbb{N}$ such that $2m + 1 \geq h$, let us consider the path π on the graph $G_h(S)$ generating the word $u = \underline{x}_{[-m,m]}$, and the paths π' and π'' given by condition $iii)$. Then there exists a configuration $\underline{y} \in C_{-m}(u)$ such that $y_i \neq x_i$ for some $i > m$.

In other words a SFT is perfect iff every finite path on the corresponding graph is extendible in at least two different ways either at the first or at the last vertex. It is sensitive to the initial conditions iff this fact holds at the last vertex of every path. As a consequence of Theorems 2 and 3, we can state that a full but non positively transitive SFT is not sensitive to the initial conditions.

Example 3. A non perfect subshift.
Let us consider the subshift of the example 1. It is not perfect since the path (0)(1) is not extendible in two different ways.

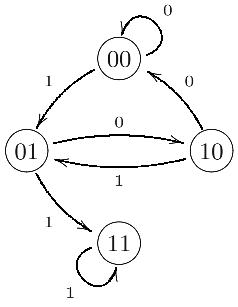


$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Fig. 3. Graph $\mathcal{AG}_{186}$ and corresponding adjacency matrix

Example 4. A sensitive subshift.
Let $\langle S_{186}, \sigma_S \rangle$ be the SFT contained in the ECA 186. Every finite path on the graph $\mathcal{AG}_{186}$ is right 2-ways extendible. Thus S_{186} is sensitive to the initial condition (and also perfect). Let us note that this SFT is not full transitive.

Example 5. A perfect subshift which is not sensitive to the initial condition.
Let $\langle S_{234}, \sigma_S \rangle$ be the SFT contained in the ECA 234. Every finite path on the graph $\mathcal{AG}_{234}$ is left 2-ways extendible but the path (01)(11) is not right 2-ways extendible. Then this SFT is perfect but not sensitive to the initial conditions.



$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Fig. 4. Graph $\mathcal{AG}_{234}$ and corresponding adjacency matrix

5 Conclusion

Given a SFT, we have shown that there exists at least one CA which contain it, and then the class of the SFT turns out to coincide with the class of SFT

Table 1. Classification of all ECA rules as left subshift.

Non Full Transitive Subshifts	
40,48,56,58,60,112,120,130,136,144,146,148,150,152 156,160,162,163,168,172,176,178,180,182,188,186,190,192,194,195 196,198,200,202,204,208,210,212,214 216,220,224,225,226 227,228,230,232,234,235,236,240,241,242,243,244,246 248,250,252	
Full Transitive and Non Positively Transitive Subshifts	
32,44,50,62,96, 104,114,122,128,131,132,134 140,154,158,161, 164,166,177,179, 203,206,218,222,233, 238,249,251,254	
Infinite Mixing Subshifts	
2,10,11,14,34,35,38,41,42,43, 46,47,57,59,66,74,98,99,106,107 138,139,142,143,155,169,170, 171,173,174,175,185,187,189,191	
Strongly Transitive Subshifts	**Strongly Transitive and Mixing Subshifts (Trivial Subshifts)**
3,15,33,45,49,51,61,63,67,75 97,105,113,115,121,123	0,4,6,8,12,16,18,20,22,24 26,28,30,36,52,54,64,68,70,72 76,78,80,82,84,86,88,90,92,94 100,102,108,110,116,118,124,126,129 133,135,137,141,145,147,149,151,153,157 159,165,167,181,183,193,197,199,201,205 207,209,211,213,215,217,219,221,223,229 231,237,239,245,247,253,255
Non Regular Subshifts	
32,35,40,44,50,58,62,96,104,105,114,122,128,130,131, 132,134,136,140,143,154 158,160,161,162,163,164,166,168,172,176,177,178, 179,186,190,202,203,206,218 ,222,224,232,233,234,235, 238,242,243,248,249,250,251,254	
Sensitive and Non Mixing Subshifts	
58,130,162,163,186	
Perfect and Non Mixing Subshifts	
40,58,130,162,163,168,172,186,202,234,235	
No Subshifts	
1,5,7,9,13,17,19,21,23,25,27,29,31,37,39,53,55,65 69,71,73,77,79,81,83,85,87,89,91,93,95,101,103, 109,111,117,119,125,127	

contained in CA. We characterized the languages associated to full transitive and sensitive subshifts. In the case of SFT, we also gave the conditions on the graph in order that the SFT exhibit these properties. The class of full but non positively transitive dynamics of subshfts (contrarily to the ones of homeomorphic CA) turns out not to be empty. Table 1 reports all the 256 ECA classified as left subshifts with respect to the full transitivity, positive transitivity, topological mixing, strongly transitivity, perfectness, sensitivity to the initial condition, and regularity. Trivially the right case can be treated in a similar way.

References

[1] J. Banks, J. Brooks, G. Cairns, G. Davis, and P. Stacey, *On Devaney's definition of chaos*, American Mathematical Montly **99** (1992), 332–334.

[2] G. Cattaneo, A. Dennunzio, and L. Margara, *Chaotic subshifts and related languages applications to one-dimensional cellular automata*, Fundamenta Informaticae **52** (2002), 39–80.

[3] M. Denker, C. Grillenberger, and K. Sigmund, *Ergodic theory on compact spaces*, Lecture Notes in Mathematics, vol. 527, Springer-Verlag, 1976.

[4] R. L. Devaney, *An introduction to chaotic dynamical systems*, second ed., Addison-Wesley, 1989.

[5] D. Lind and B. Marcus, *An introduction to symbolic dynamics and coding*, Cambidge University Press, 1995.

[6] P. Walters, *An introduction to ergodic theory*, Springer, Berlin, 1982.

Evolution and Observation: A Non-standard Way to Accept Formal Languages*

Matteo Cavaliere[1] and Peter Leupold[2]

[1] Department of Computer Science and Artificial Intelligence
University of Sevilla,
Sevilla, Spain
martew@inwind.it

[2] Research Group in Mathematical Linguistics
Rovira i Virgili University
Tarragona, Spain
klauspeter.leupold@estudiants.urv.es

Abstract. It is a very common procedure in biology to observe the progress of an experiment and regard the result of this observation as the final outcome. Inspired by this, a new approach for generating formal languages, called evolution/observation, has been introduced [6]. In the current work we consider evolution/observation as a new strategy also for accepting languages: a word is accepted, if the (observed) evolution of a certain system starting from this input follows a regular pattern. We obtain the following result: checking if the (observed) evolution of a context-free system follows a regular pattern is enough to accept every recursively enumerable languages. On the other hand, if we observe the evolution of systems using very simple rules (of the kind $a \to b$), then it is possible to accept exactly the class of context-sensitive languages.

Keywords: Evolution, Observation, Languages, Universality.

1 Introduction: Using Evolution/Observation to Accept Languages

In earlier work of the current authors, a new approach for generating languages, called *evolution/observation* was introduced [5]; the idea comes from the fact that often, in biology and chemistry, the result (i.e. the output) of a certain experiment is the observation of the entire progress of the experiment.

In the mentioned work, it has been shown how a language-generating device can be constructed using two less powerful systems: a mathematical model of a biological system that "lives" (evolves), and an observer that watches the entire evolution of the biological system and translates it into a readable output. Thus

* This work was done, while both authors were funded by the Spanish Ministry of Culture, Education and Sport under the Programa Nacional de Formación de Profesorado Universitario (FPU).

the main idea of this approach is that the computation is made by observing the entire "life" of a biological system.

This architecture has already been applied in different frameworks. In a first article [5] the evolution of a *membrane system* (a formal model inspired by the functioning of the living cells) was observed. There it was shown that a system composed of a membrane system with only context-free power and a finite state automaton in the role of the observer is computationally universal. This can be considered a first hint towards the fact that such an approach is powerful.

In subsequent work [4], a finite automaton observed the evolution of "marked" strings of a *splicing system* (a formal system inspired by the recombination of DNA strands that happens under the action of restriction enzymes). Also in this case, the observation adds much power to the considered bio-system. In particular, it has been shown that just observing the evolution of "marked" strings in a splicing system (using finite axioms and rules) it is even possible to generate all the context-free languages; the generative power of this class of splicing systems, considered in the standard way, is subregular.

In another approach [1], the evolution/observation strategy has been applied to *sticker systems* (a formal model inspired by the ligation and annealing operation largely used in the DNA computing area). As now somehow expected, also in this case just observing the evolution of the double strand DNA molecules obtained using the simple regular sticker operation, it is possible to generate non-recursive languages; the classical generative power of simple regular sticker systems is subregular.

Finally, the evolution/observation framework has also been used, in a more general way, for generating formal languages: the "evolution" of a *grammar* was observed using a finite automaton. In this case, universality is obtained using a finite state automaton observing a context-free grammar [6]. Also some non-universal variants have been presented.

In distinction to all the previous works, where the strategy of evolution/observation has been used to construct devices generating languages, here we want to use this strategy to construct a device that accepts languages. This will be called a *recognizing E-system/observer*.

Such a recognizer E-system/observer is composed of three components: an evolving system, in short *E-system*, an *observer* and a *decider*. Figure 1 schematically illustrates the manner in which these three components interact to form a system.

When a word w is given as input to a recognizer E-system/observer, then the E-system starts to modify this word according to its rules. In this way it generates a sequence of *intermediate words*. The E-system stops, when the last word obtained cannot be rewritten any more. The entire sequence is regarded as the *behavior* of the E-system when it receives as input the original word w. The observer, following a set of specific rules, associates a label to each intermadiate word of this behavior. It writes these labels onto an output tape in their chronological order, and in this way a word w' is obtained, which describes the behavior observed.

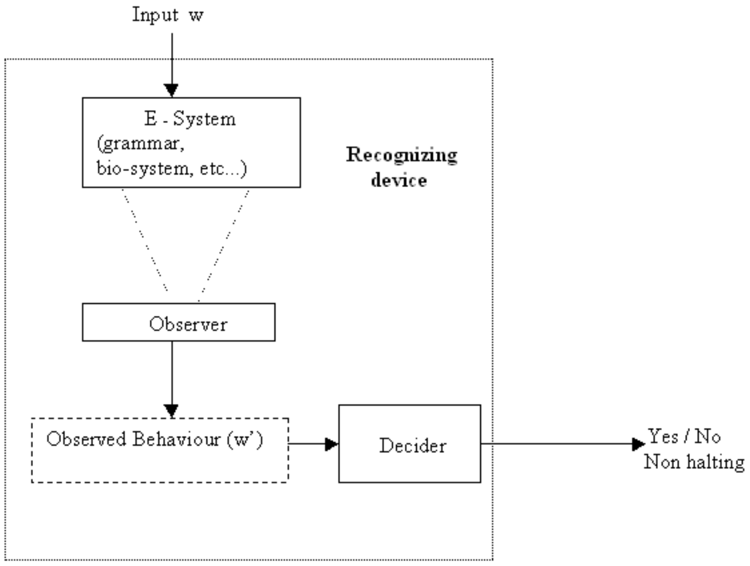


Fig. 1. The architecture of a recognizing E-system/observer.

Then the original word w is accepted, if and only if w' is in the regular language accepted by the decider. In other words, the word w is accepted, if and only if the observed behavior of the E-system with w as input has followed a certain regular pattern.

Coming back to the original motivation of a recognizing E-system/ observer we can imagine using a biological system as a recognizing device: just taking the biological system, "introducing" an input to such a system and observing its evolution. If the evolution of the biological system is the one expected (for example follows a regular pattern) then the word is accepted by the system, otherwise it can be considered rejected. The observer is fundamental in extracting a more abstract formal behavior from the evolution of the biological system, somewhat like a protocol of the evolution. The decider checks that the behavior of the biological system was the one expected.

The E-system used can be any system where some kind of behavior can be formalized: a rewriting system or the mathematical model of some biological system like membrane, splicing, sticker systems, etc.

We want to stress the main difference with classical accepting devices and with grammars used as accepting devices as introduced by Bordihn et al. [3]: in our case the acceptance of a word is decided *analyzing the entire life* of the considered E-system.

Further we have to remark some similarities of our approach with the idea of *iterated gsm*. Also there, a initial word is rewritten and in this way a language is generated. However, in every step one word is generated, see for instance [9], [10], whereas here the entire iteration accepts only one single word. Except for the way of defining the language, our device can be seen like a case of an iterated (accepting) gsm, with a regularly controlled sequence of transductions. A similar kind of device has been studied by Asveld [2].

Another somewhat comparable mechanism is given by Ilie and Salomaa [8] in a characterization of recursively enumerable languages. There two systems of rules –just like the ones used below– alternate according to a regular control mechanism, and this way computational universality is achieved.

In what follows we suppose the reader familiar with the basics of formal languages. In general, for all notions from formal language theory we refer to standard textbooks (e.g. [11] and [12]). By CF, CS, and RE we denote the classes of languages generated by context-free, context-sensitive, and unrestricted grammars respectively.

2 Recognizing G-system/Observer: Definition

In this section we define formally a *recognizing G-system/observer* as a particular instance of the general E-system/Observer architecture presented in the introduction. We start out by providing separately the definitions of the three components that are then used to form a recognizing G-system/observer.

2.1 G-systems

As underlying E-system effecting the rewriting (evolution) of the input string, we will consider a simplification of generative grammars. There is no start symbol, and like in *pure grammars* there is no distinction made between terminals and non-terminals. Thus such a *G-system* is a pair $\Gamma = (V, P)$, where V is a finite alphabet, and P is a finite set of rewriting rules over V.

The system Γ rewrites an input word $w \in V^*$ in exactly the same way as a grammar during a derivation. As such a derivation is in general non-deterministic, also the G-system Γ can produce many different sequences of intermediate strings (the equivalent of sentential forms for grammars); such sequences have the form $\langle w_1, w_2, \ldots, w_n, \ldots \rangle$ for words $w_i \in V^*$ and $w_1 = w$. The set of all such sequences for which a system Γ on a word w finally stops – i.e. no more rule can be applied – is denoted by $\Gamma(w)$.

If all the rules in P are context-free, then Γ is called a CF G-system. The class of all such systems will be shortly denoted by $\mathcal{CF}_{\mathcal{GS}}$. We also consider a more restricted case of G-systems having only rules of the kind $a \to b$, for $a, b \in V$. It is interesting to observe that these rules are not only regular, but even changing only one letter into another one. Thus the length of the string rewritten by the G-system is constant, and in some sense the system is just modifying it to investigate its structure. Because one might also see this as just repainting the letters in different colors, we will call such set of rules a *painter* and denote the class of all G-systems using only painters by $\mathcal{PA}$.

2.2 Observer

For the observer as described in the introduction, however, we need a device mapping arbitrarily long strings into just one singular symbol. As in earlier work [6] we use a special variant of finite automata with some feature known from Moore machines: the set of states is labelled with the symbols of an output alphabet Σ. Any computation of the automaton produces as output the label of the state it halts in (we are not interested in accepting / not accepting computations and therefore also not interested in the presence of final states); because the observation of a certain string should always lead to a fixed result, we consider here only deterministic and complete automata.

Formalizing this, a *monadic transducer*[3] is a tuple $O = (Z, V, \Sigma, z_0, \delta, \sigma)$ with state set Z, input alphabet V, initial state $z_0 \in Z$, and a complete transition function δ as known from conventional finite automata; further there is the output alphabet Σ and a labelling function $\sigma : Z \mapsto \Sigma$. The output of the monadic transducer is the label of the state it stops in. For a string $w \in V^*$ and a transducer O we then write $O(w)$ for this output; for a sequence $\langle w_1, \dots, w_n \rangle$ of $n \geq 1$ strings over V^* we write $O(w_1, \dots, w_n)$ for the string $O(w_1) \cdots O(w_n)$. The class of all (deterministic) monadic transducers will be denoted by $\mathcal{MT}$. For simplicity, in what follows, we present only the mappings that the observers define, without giving detailed implementations for them.

2.3 Decider

As *deciders* we require devices accepting a certain language over the output alphabet Σ of the corresponding observer as just introduced. For this we do not need any new type of device but can rely on conventional finite automata with input alphabet Σ. The output of the decider, for a word $w \in \Sigma^*$ in input, is denoted by $D(w)$. It consists in a simple `yes` or `no`. The class of all deciders will be denoted by $\mathcal{FA}$, like for standard finite state automata.

2.4 Recognizing G-system/Observer

Putting together the components just defined in the way informally described in the introduction, a recognizing G-system/observer (in short *RGO*) is a quadruple $\Omega = (\Delta, \Gamma, O, D)$; here Δ is the finite input alphabet, $\Gamma = (V, P)$ is a G-system where $\Delta \subseteq V$, O is an observer $(Z, V, \Sigma, z_0, \delta, \sigma)$, and D is a decider with input alphabet Σ.

The language accepted by such a system is the set of all words $w \in \Delta^*$ such that there exists a sequence $s \in \Gamma(w)$ such that $D(O(s)) = \texttt{yes}$; formally

$$L(\Omega) := \{w : \exists s \in \Gamma(w)[D(O(s)) = \texttt{yes}]\}.$$

[3] In earlier work these devices were called *finite automata with singular output*. We introduce here this new name inspired by monadic rewriting systems, because it seems much less awkward.

For a class $\mathcal{G}$ of G-systems, $\mathcal{O}$ of observers and $\mathcal{D}$ of deciders, we denote by $\mathcal{RGO}(\mathcal{G}, \mathcal{O}, \mathcal{D})$ the class of all languages accepted by *RGO*s using components from the respective classes.

3 The Power of *RGO*s Using Painters

In this section we study the accepting power of *RGO*s using restricted G-systems having only painter rules. We prove that this type of *RGO* is able to accept exactly the class of context-sensitive languages. Initially, we illustrate the manner in which a RGO works by giving an example. It shows how the combination evolution/observation of very simple components can be used to accept more complicated languages.

Example 3.1 The non context-free language $L' = \{a^n b^n c^n : n \in \mathbb{N}\}$ can be accepted by an *RGO* $\Omega = (\Delta, \Gamma, O, D)$, with the following components:
the *input alphabet* Δ is $\{a, b, c\}$;
the *G-system* $\Gamma = (V, P)$ has alphabet
$V = \{a, b, c, a', a'', b', b'', c', c''\}$ and the set P of rules (painters) is

$$\{a \to a', a' \to a'', b \to b', b' \to b'', c \to c', c' \to c''\};$$

the *observer* O, with input alphabet V and output alphabet
$\Sigma = \{a_1, a_2, a_3, a_4, a_5, a_6, \perp\}$, realizes the following mapping:

$$O(w) = \begin{cases} a_1 & \text{if } w \in a''^* a' a^* b''^* b^* c''^* c^+, \\ a_2 & \text{if } w \in a''^* a' a^* b''^* b' b^* c''^* c^*, \\ a_3 & \text{if } w \in a''^* a' a^* b''^* b' b^* c''^* c' c^*, \\ a_4 & \text{if } w \in a''^* a^* b''^* b' b^* c''^* c' c^*, \\ a_5 & \text{if } w \in a''^* a^* b''^* b^* c''^* c' c^*, \\ a_6 & \text{if } w \in a''^* a^* b''^* b^* c''^* c^*, \\ \perp & \text{else.} \end{cases}$$

The *decider* D is a finite state automaton, with input alphabet Σ, that gives a positive answer exactly if a word belongs to the regular language

$$a_6(a_1 a_2 a_3 a_4 a_5 a_6)^*;$$

note that in this way any production of the symbol $\perp$ leads to immediate rejection of the word.

What are now the input words w over V^*, for which there exists at least one sequence s in $\Gamma(w)$ such that $O(s)$ belongs to the regular language accepted by the decider D? We start by considering $w = \lambda$; obviously $\Gamma(w)$ contains only the sequence $< \lambda >$. Because we have $O(< \lambda >) = a_6$, then $O(< \lambda >)$ is accepted by the decider and therefore the empty word is accepted by the entire system Ω.

Consider next a non-empty word w; first, we notice that such a word must belong to $a^+b^+c^+$, otherwise there will be no sequence s in $\Gamma(w)$ such that $O(\Gamma(w)) = w'$ and w' starts with the necessary a_6.

If this is true, then always the six stages corresponding to a_1 to a_6 can be traversed. Every time exactly one of each of the letters a, b, c in w is transformed, using the rules in P, to its doubly primed version. If we arrive, in this manner, at some word from $a''^+b''^+c''^+$, then we have obtained a sequence $s \in \Gamma(w)$ of intermediate strings such that $O(s)$ is accepted by the decider and then the word w is accepted by Ω; the regular expression checked by the decider guarantees that the word w was of the form $a^+b^+c^+$ and had exactly the same number of as, bs and cs.

In fact, for any word w that does not respect this structure, the sequences in $\Gamma(w)$ will correspond to strings different from the one accepted by the decider; for example if in w there are more bs than as and cs then every sequence s in $\Gamma(w)$ will be such that $O(s)$ contains more a_2s than a_1s and a_3s, and thus rejected by the decider. Therefore the language accepted by Ω is exactly the language L'.

We now try to determine the exact power of this type of *RGO*. It will turn out to be equal to that of Linear Bounded Automata; as these are defined in terms of complexity theory, we will use several notions from that field without introducing them in detail. For a language L to be in $NSPACE-TIME(f,g)$ means that there exists a non-deterministic Turing Machine accepting L and not using more than $f(n)$ tape-cells in $g(n)$ steps to accept a word of length n. For details about these and all other notions concerning complexity we refer the reader to the monograph of Wagner and Wechsung [13]. To denote anonymous functions we use the notation of the lambda-calculus; thus, for example, $\lambda n.n^2$ is the square function with argument n. By *id* we denote the identity function $\lambda n.n$.

We now show that for the work space there is a very tight and obviously optimal bound telling us that $RGO(\mathcal{PA}, \mathcal{MT}, \mathcal{FA}) \subseteq CS$.

Proposition 3.2 *Any language L from the class $RGO(\mathcal{PA}, \mathcal{MT}, \mathcal{FA})$ lies in the class $NSPACE-TIME(id, \lambda n.n \cdot c^n)$, where c is the size of the corresponding G-system's alphabet.*

Proof. With a non-deterministic Turing machine with one tape we simulate a *RGO* using a G-system $\Gamma \in \mathcal{PA}$, observer $Ob \in \mathcal{MT}$ and decider $D \in \mathcal{FA}$, in the following way: the actions of Γ are simulated by rewriting symbols only on the positions of the tape originally occupied by the input word. This takes at most $n-1$ steps to reach the (randomly chosen) position in question. Then *Ob* is simulated in the machine's finite control; nothing is written on the tape, in the worst case we need $n-1$ steps to reach the first position of the word, then n steps to simulate *Ob*.

Now *Ob*'s single-letter output is not written on the tape – rather in the control we remember D's last state and change it according to the letter *Ob* would output. In this way none of the output need ever be stored, because it is generated from left to right, just the way D reads it – therefore the reading can be simulated "on-line" rather than doing it after everything else is finished. All the phases of this process take a number of steps linear in n, and further we can already see that the proposition's space bound holds.

For the time bound we need some more considerations. Let c be the size of Γ's alphabet. As the length of the input word is constant, from a word of length n at most c^n different words can be reached via a painter's rules. Further, looking at the simulation of a step of the original system at the point where the Turing Machine's head is on the tape's first position ready to simulate Ob, this type of configuration is recurring for every step, because the position of the head is determined, and in the control only the last state of D need be stored. So if D has k states, the number of distinct configurations of this type is $k \cdot c^n$. Any computation simulating more steps of the RGO contains a cycle and therefore has the same result as a shorter one (if it terminates).

Since, as already stated above, the number of Turing machine steps to simulate one RGO step is linear in n, the overall shortest accepting computation (if it exists) accepting a word of length n has a time bound in the order of $\lambda n.n \cdot c^n$. □

The time bound we obtain from this simulation is not optimal. With the following theorem and the space bound from Proposition 3.2 we can see that $RGO(\mathcal{PA}, \mathcal{MT}, \mathcal{FA}) \subseteq NSPACE - TIME(id, \lambda n.2^n)$ must hold. It should be noted that in the cited theorem a somewhat different version of Turing Machines is used (off-line there vs. on-line here); in our case, however, the result carries over to the type of machine described in the proof of Proposition 3.2.

Theorem 3.3 (see e.g. [13]) *Let $s \geq \log$ be a space-constructible function; then $NSPACE(s) \subseteq NSPACE - TIME(s, 2^s) \subseteq NTIME(2^s)$.*

Now we give a lower bound for $RGO(\mathcal{PA}, \mathcal{MT}, \mathcal{FA})$'s space complexity by showing that any language accepted by a Linear Bounded Automaton is in this class.

Proposition 3.4 *$NSPACE(id) \subseteq RGO(\mathcal{PA}, \mathcal{MT}, \mathcal{FA})$.*

Proof. We construct for any LBA M an equivalent system Ω with a G-system from $\mathcal{PA}$, an observer from $\mathcal{MT}$, and a decider from $\mathcal{FA}$. The LBAs are normed in the following manner: the input word's left border is signified by a #, the right border by a \$; from these symbols no transition moves to the left, respectively to the right. The set of transitions δ is composed of elements of the form $Q \times A \rightarrow Q \times A \times \{+, -\}$, where Q is the set of states, A the tape alphabet, and $+$ or $-$ denotes a move to the right or left respectively. An input word is accepted, if the LBA stops in a final state.

Finally, we take as input to Ω not words w, but their bordered version $\#w\$$; this makes simulation a little easier, but does not restrict generality: one may as well code # and w's first letter into one letter and replace this first letter by the new one before anything else is done; then by standard techniques from complexity theory we simulate the machine's action on both positions in the single one. On the right border the same is true. This, however, would make Ω much more complicated.

Now we build up a $RGO\ \Omega = (A, \Gamma, O, D)$ simulating M. Here $\Gamma = (V, P)$ where the alphabet $V = A \cup \{\#, \$\} \cup (A \times Q) \cup T$. All transitions in δ have associated an unique label t, and T is the set of all these labels. A letter from the set $A \times Q$ shall indicate that M's head is in the indicated position and the current state is the component from Q. To facilitate any potential configurations the rule set P contains all possible rules of the form $x \to y$, where $x \in A$ and $y \in \{x\} \times Q$.

Further, for all transitions $t : (q_1, x) \to (q_2, y, \mu)$ the rules $(x, q_1) \to t$ and $t \to y$ are added to P. We give now an example of how such transition's simulation works for $\mu = +$; letters from $A \times Q$ with two components are depicted in the way $\frac{\text{component1}}{\text{component2}}$, a letter a from A by $\frac{a}{-}$:

$$\frac{x}{q_1}\frac{z}{-} \overset{(x,q_1)\to t}{\Longrightarrow} \frac{t}{-}\frac{z}{-} \overset{z\to(z,q_2)}{\Longrightarrow} \frac{t}{-}\frac{z}{q_2} \overset{t\to y}{\Longrightarrow} \frac{y}{-}\frac{z}{q_2}$$

The rest of the word should consist exclusively of letters from A. For the simulation of each transition $t : (q_1, x) \to (q_2, y, +)$, the mapping realized by the observer with output alphabet $\{a, b, c, d, e\}$ is the following:

$$O(w) = \begin{cases} a & \text{if } w \in \#A^*(x, q_1)A^*\$ \cup (\#, q_1)A^*\$, \\ b & \text{if } w \in \#A^*tA^*\$ \cup tA^*\$, \\ c \text{ or } d & \text{if } w \in \#A^*t(y, q_2)A^*\$ \cup t(y, q_2)A^*\$ \cup \#A^*t(\$, q_2) \cup t(\$, q_2). \end{cases}$$

The configuration with $\frac{y}{-}\frac{z}{q_2}$ is again of the first type. The latter parts of all expressions are necessary only for the special cases, where the head touches one of the borders. Transitions moving to the left ($\mu = -$) are treated analogously. In the last clause, c is output if q_2 is an accepting state, d otherwise. The initial configuration $\#w\$$ is mapped to a. In all configurations not mentioned here, the output is e.

The final component of Ω, the decider, accepts the language

$$a(ab(c \vee d))^*abc.$$

It should be noted that, due to the uniqueness of each transition t, the clauses for b and c/d are unique for each one, too; therefore there is no ambiguity in the observer. As the symbol t is produced only by an appropriate letter/state combination (x, q_1), every sequence abc or abd produced by the observer corresponds to the correct simulation of the transition t. On the other hand, such sequences cannot be produced in any other way. Thus the language accepted is exactly the one accepted by the original M. □

Summarizing Propositions 3.2 and 3.4 we now see that actually the class $RGO(\mathcal{PA}, \mathcal{MT}, \mathcal{FA})$ is equal to a well-known language class, as $NSPACE(id)$ is exactly the class of context-sensitive languages.

Theorem 3.5 $RGO(\mathcal{PA}, \mathcal{MT}, \mathcal{FA}) = CS$.

4 The Power of *RGO*s Using Context-Free Rules

Considering *RGO*s with regular G-system does not make sense here: as the rules do not distinguish between terminals and non-terminals, there is no difference between context-free and regular rules. Therefore we proceed directly to the investigation of CF G-systems. Here we already reach the maximal possible power, because all recursively enumerable languages can be accepted.

Theorem 4.1 $RGO(\mathcal{CF}_{\mathcal{GS}}, \mathcal{MT}, \mathcal{FA}) = RE.$

Proof (Sketch). We sketch, for any given Turing Machine M, the construction of an equivalent RGO Ω. This is done completely analogously to the proof of Proposition 3.4. We only need to provide the work tape with an unbounded amount of space potentially used by M in contrast to an LBA. This is done by adding to the G-system of Ω a special letter $\Box$, and the context-free rules $x \to x'$, $x' \to \Box x'$ $x' \to x'\Box$, and $x' \to x$ for all letters x from M's input alphabet. These can expand the tape to both sides of the input word.

This way any amount of work space can be produced before starting the simulation of a computation. The priming of the letter ensures that –with appropriate design of the observer– space is generated only before the computation's simulation. In general, later generation would not be a problem, only in cases where the machine's head is on a blank symbol and more space is introduced between the head and the non-empty part of the tape. $\Box$

5 Outlook

Interesting problems have been left open: the *RGO*s considered accept languages in a highly non-deterministic manner that makes the recognizing systems not useful from a practical point of view; this raises the question of how to decrease the non-determinism. One option is to consider a type of *RGO*s, *confluent* in the sense that for a word w all possible computations result in the same answer.

Finally, as already mentioned in the introduction, it seems especially interesting for the possible practical relevance, to consider *RGO*s where the G-system used is the mathematical model of some biological system. For instance, in the DNA area, there has been introduced a lab-technique called FRET [7], used to observe, in real-time, the dynamic of DNA strands. This technique might be used to implement for a DNA computation the type of observer introduced in our framework.

References

1. A. Alhazov and M. Cavaliere: Computing by Observing Bio-Systems: the Case of Sticker Systems, Pre-Proceedings of DNA 10 - Tenth International Meeting on DNA Computing, Milano, 2004.
2. P. R. J. Asveld, On controlled iterated gsm mappings and related operations, *Preprint Math. Centrum*, Amsterdam, 1979, *Rev. Roum. Math. Pures.*

3. H. Bordihn, H. Fernau and M. Holzer: Accepting Pure Grammars and Systems. Technical Report 1, Fakultät für Informatik, Universität Magdeburg, 1999.
4. M. Cavaliere and N. Jonoska: (Computing by) Observing Splicing Systems. Manuscript 2004.
5. M. Cavaliere and P. Leupold: Evolution and Observation – A New Way to Look at Membrane Systems. In: C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa (eds.): Membrane Computing, Lecture Notes in Computer Science 2933, Springer, 2004, 70–88.
6. M. Cavaliere and P. Leupold: Evolution and Observation — A Non-Standard Way to Generate Formal Languages. *Theoretical Computer Science* 321, 2004, pp. 233-248.
7. T. Ha, Single-Molecule Fluorescence Resonance Energy Transfer, *Methods*, 25, 2001, 78–86.
8. L. Ilie and A. Salomaa: 2-Testability and Relabelings Produce Everything. *Journal of Computer and System Sciences* 56(3), 1998, pp. 253-262.
9. V. Manca, C. Martin-Vide, Gh. Păun, Iterated GSM mappings: A collapsing hierarchy, *Jewels are Forever* (J. Karhumaki, H. Maurer, Gh. Păun, G. Rozenberg, eds.), Springer-Verlag, 1999, 182–193.
10. Gh. Păun, On the iteration of gsm mappings, *Rev. Roum. Math. Pures Appl.*, 23, 4 (1978), 921–937.
11. G. Rozenberg, A. Salomaa (eds.): *Handbook of Formal Languages.* Springer-Verlag, Berlin, 1997.
12. A. Salomaa: *Formal Languages.* Academic Press, New York, 1973.
13. K. Wagner and G. Wechsung: *Computational Complexity.* Deutscher Verlag der Wissenschaften, Berlin, 1986.

The Computational Power of Continuous Dynamic Systems

Jerzy Mycka[1]* and José Félix Costa[2]

[1] Institute of Mathematics,
University of Maria Curie-Sklodowska
Lublin, Poland
Jerzy.Mycka@umcs.lublin.pl

[2] Department of Mathematics, I.S.T.
Universidade Técnica de Lisboa
Lisboa, Portugal
fgc@math.ist.utl.pt

Abstract. In this paper we show how to explore the classical theory of computability using the tools of Analysis: a differential scheme is substituted for the classical recurrence scheme and a limit operator is substituted for the classical minimalization. We show that most relevant problems of computability over the non negative integers can be dealt with over the reals: elementary functions are computable, Turing machines can be simulated, the hierarchy of non computable functions be represented (being the classical halting problem solvable in some level). The most typical concepts in Analysis become natural in this framework.

1 Introduction and Motivation

The theory of analog computation, where the internal states of a computer are continuous rather than discrete, has enjoyed a recent resurgence of interest. This stems partly from a wider program of exploring alternative approaches to computation, such as neural and quantum computation; partly as an idealization of numerical algorithms where real numbers can be thought of as entities in themselves, rather than as strings of digits [19]; and partly from a desire to use the tools of computation theory to better classify the variety of continuous dynamical systems that model our world (or at least its classical idealization) [3,9,18]. If we are to make the state of a computer evolve in a continuum it makes sense to consider making its progress in time continuous too. While a few efforts have been made in the direction of studying computation by continuous-time dynamical systems [9,13,18,1], no particular set of definitions has become widely accepted. Thus analog computation has not yet experienced the unification that digital computation did through Turing's work in 1936.

In this work we go back to the roots of analog computation theory by starting with Claude Shannon's so-called General Purpose Analog Computer (GPAC).

* Corresponding author

This was defined as a mathematical model of an analog device, the Differential Analyser, the fundamental principles of which were described by Lord Kelvin in 1876 [8]. Just as polynomial operations are basic to the Blum-Shub-Smale (BSS) model of analog computation [2], polynomial differential equations are basic to the GPAC. Rubel [16] proposed the Extended Analog Computer (EAC). This model has the same computational power as the GPAC but also produces the solutions of a broad class of Dirichlet boundary-value problems for partial differential equations.

The first presentation of a Theory of Recursive Functions over the Reals was attempted by Cris Moore [9]. Real recursive functions are generated by a fundamental operator, called differential recursion. The other fundamental operator is the taking of limits [11]. Between 1996, since Moore's seminal paper, and 2002 we have been working with the single concept of differential recursion. In [6] we show that a linearizarion of the differential recursion scheme gives rise to an analog characterization of the class of (Kalmar's) elementary functions. In [5] and [7] we show that the GPAC is not closed under iteration and that a subclass of real recursive functions coincides with the class of GPAC-computable functions. In [11] we finally show how to capture higher computational classes through the limit operator. Manuel Campagnolo in [4] showed also that other computational complexity classes can be captured through appropriate structured differential schemata or adding simple (bounded) integration. About the hierarchy of limits, we may add further topics. We show that we can embed the entire arithmetical hierarchy within the limit hierarchy up to some finite level (up to a finite number of limit operations), where the analytic hierarchy starts to be implemented. The use of limits gives rise to uncomputable functions, e. g., at some level we get the halting problem solved. To these previous aspects, we should add the impact of a further one: in the basis of the limit hierarchy we can still find a set of functions over the reals indeed computable by physical means, theoretically by Claude Shannon's GPAC and practically by the Differential Analyser of Vannevar Bush. Hence, in the basis we have truly computable functions in the physical sense. Can we envisage engines with a greater power? A final remark helps the reader to understand that computable numbers can be thought as entire computable structures, indivisible entities [9] or computable by digits (as in the classical way), using continued fractions. Strong uncompressible numbers like Chaitin's halting probability are computable in very precise levels of the limit hierarchy.

Now we finish by recalling to the reader Moore's seminal paper [9] published in 1996. Herein, we try to reformulate many of his constructs that failed to have a strong foundational basis. Reimplementation of the arithmetical hierarchy, and analytical hierarchy by means of continued fractions, are included in this paper in his honour, just to say that after all every construct in [9] can still stand up on top of our hierarchy of limits.

2 Recursive Functions over the Reals with Bounded Differential Recursion and Infinite Limits

We give a new definition of real recursive functions, which is a derivative of the original definition found in [9].

However it is invented to avoid problems involved in the latter. It is important to see that the following definition is based on the vector operations.

Definition 2.1. *The set of real recursive vectors is generated from the real recursive scalars* $0, 1, -1$ *and the real recursive projections* $I_n^i(x_1, \ldots, x_n) = x_i, 1 \leq i \leq n, n > 0$*, by the operators:*

1. *composition: if f is a real recursive vector with n k-ary components and g is a real recursive vector with k m-ary components, then the vector with n m-ary components ($1 \leq i \leq n$)*

$$\lambda x_1 \ldots x_m . f_i(g_1(x_1, \ldots, x_m), \ldots, g_k(x_1, \ldots, x_m))$$

   is real recursive.

2. *differential recursion: if f is a real recursive vector with n k-ary components and g is a real recursive vector with n $k+n+1$-ary components, then the vector h of n $k+1$-ary components which is the solution of the Cauchy problem for $1 \leq i \leq n$*

$$h_i(x_1, \ldots, x_k, 0) = f_i(x_1, \ldots, x_k),$$

$$\partial_y h_i(x_1, \ldots, x_k, y) = g_i(x_1, \ldots, x_k, y, h_1(x_1, \ldots, x_k, y), \ldots, h_n(x_1, \ldots, x_k, y))$$

   is real recursive whenever h is of the class C^1 on the largest interval containing 0 in which a unique solution exists.

3. *infinite limits: if f is a real recursive vector with n $k+1$-ary components, then the vectors h, h', h'' with n k-ary components ($1 \leq i \leq n$)*

$$h_i(x_1, \ldots, x_k) = \lim_{y \to \infty} f_i(x_1, \ldots, x_k, y),$$

$$h'_i(x_1, \ldots, x_k) = \liminf_{y \to \infty} f_i(x_1, \ldots, x_k, y),$$

$$h''_i(x_1, \ldots, x_k) = \limsup_{y \to \infty} f_i(x_1, \ldots, x_k, y)$$

   are real recursive, whenever these limits are defined for all $1 \leq i \leq n$.

4. *Arbitrary real recursive vectors can be defined by assembling scalar real recursive components of the same arity.*
5. *If f is a real recursive vector, than each of its components is a real recursive scalar.*

Because every function has at least one finite syntactical description, hence the number of real recursive functions is countable. In this way we can observe that the system of functions given by our definition is constructive and not too large.

Let us discuss carefully the details of the definition. For differential recursion we restrict a domain to an interval of continuity. This operator gives the same class C^k for a defined function as the given functions are from. This eliminates the possibility of defining such functions as $\lambda x.|x|$ without the limit operator. We excluded here the possibility of operations on undefined functions: our functions are strict in the meaning that for undefined arguments they are also undefined. But to obtain some interesting functions we should improve the power of our system by an addition of the operators of infinite limits. Let us point out that introducing of infinite limits gets discontinuous functions.

Following [20] (chapter 3), consider a real recursive function f such that: (i) f is continuous on $[0, \infty)$ except possibly for a finite number of jump discontinuities in every finite subinterval; (ii) there is a positive number M such that $|f(t)| \leq Me^{kt}$ for all $t \geq 0$. Then we say that f belongs to the class L_k. Additionally let $L = \bigcup_{k>0} L_k$.

Proposition 2.1. *If $f \in L_k$, then the Laplace transform $L(f)(x)$ exists for $x > k$ and it is real recursive.*

Proof. From the condition (ii) we have $|f(t)|e^{-kt} \leq M$. Now we can proceed in the following way:

$$\int_0^y |f(t)|e^{-xt}dt = \int_0^y |f(t)|e^{-kt}e^{-(x-k)t}dt \leq \int_0^y Me^{-(x-k)t}dt.$$

Now to check an existance of the Laplace transform it is sufficient to take $\lim_{y\to\infty} \int_0^y Me^{-(x-k)t}dt$ which is defined only if $x > k$, and in this case is given by $\frac{M}{x-k}$. □

If the Laplace transform of f exists, then f is said to be of exponential order: it exists for x greater than some real number k. If the Laplace transform of f exists, and $L(f)(s)$ is defined for $s > 0$, then f is of subexponential order.

We can also present the proposition, which connects inverse Laplace transform with real recursive functions.

Proposition 2.2. *Let F, G be Laplace transforms of some real recursive functions. Then inverse Laplace transforms $L^{-1}(FG), L^{-1}(F + G)$ are also real recursive functions.*

The above proposition is a consequence of properties of inverse Laplace transform, namely convolution and linearity.

To illustrate further this transformation let us point out that if f is a $n+1$-ary real recursive function, then its derivative $\partial y f(x_1, \ldots, x_n, y)$ is a real recursive function. This result can be obtained directly by limits or for some functions by properties of Laplace transform. For example, let $f(x) = x^n, n \geq 0$, then $L(f)(s) = \frac{n!}{s^{n+1}}$. By the known property of Laplace transform we have: $L(\partial_x f(x))(s) = sL(f(x)) - f(0) = \frac{n!}{s^n}$, using repeatedly the convolution operation for $\frac{1}{s}$ we get $L^{-1}(\frac{n!}{s^n})(x) = nx^{n-1}$. Derivatives are physical realizable:

the class of differential algebraic functions is closed under derivatives, making a large class of derivatives physical realizable. Let us give without a proof some examples of functions generated with the definition of real recursive functions.

Proposition 2.3. *The functions* $+, \times, -, \exp, \sin, \cos, \lambda x.\frac{1}{x}, /, \ln, \lambda xy.x^y$, *the Kronecker* δ *function, the signum function, and absolute value are real recursive functions. The Heaviside* Θ *function (equal to 1 if* $x \geq 0$, *otherwise 0), the binary maximum* max, *the square-wave function, and the floor function* $\lambda x.\lfloor x \rfloor$ *are all real recursive too.*

Because the set of natural numbers can be defined by real recursive functions, hence we can extend the definition of real recursive numbers for functions with a domain in $N^k \times R^n, k, n \geq 0$, by the following method: *a function* $f : N^k \times R^n \to R^m, n \geq 0$ *is called real recursive if* $f(n_1, \ldots, n_k, y_1, \ldots, y_n) = F(n_1, \ldots, n_k, y_1, \ldots, y_n)$, *for some real recursive function* $F : R^{k+n} \to R^m$.

We gave the general definition of real recursive functions. For proper analysis of functions it is important to control the domain and singularities of functions. We can postulate new operators which may check the points: are they in the domain of some functions or not. For any function $f : R^{n+1} \to R$ let $\eta_y f(\bar{x}, y) = \begin{cases} 1 \text{ if } \lim_{y\to\infty} f(\bar{x}, y) \text{ is defined,} \\ 0 \text{ otherwise.} \end{cases}$ In the analogous way we can define $\eta^i_y f(\bar{x}, y), \eta^s_y f(\bar{x}, y)$ equal to 1 if $\liminf_{y\to\infty} f(\bar{x}, y)$ is defined (resp. $\limsup$), otherwise equal 0. The below proposition is cited after [11].

Proposition 2.4. *The functions* $\eta_y g, \eta^i_y g, \eta^s_y g$ *are total real recursive functions if* g *is total real recursive function.*

An iteration of a given function plays the important role in computability theory. For given function $h(x)$ we can built the consecutive values:

$$x, h(x), h(h(x)), \ldots, \underbrace{h(\ldots(h(x))\ldots)}_{k}, \ldots.$$

They are usually denoted by $h^k(x)$. We can present the result (cf. [11]) about a possibility of such construction in the set of real recursive functions.

Proposition 2.5. *Let* $h : R \to R$ *be a real recursive function. Then the function* $H : N \times R \to R$, $H(n, x) = h^n(x)$ *is real recursive too.*

The above result can be easily extended for vectors.

As a corollary we can obtain the fact that for a real recursive function $f : R^{n+1} \to R$ its finite product $F_1(n, \bar{x}) = \prod_{i=0}^n f(i, \bar{x})$ and finite sum $F_2(n, \bar{x}) = \sum_{i=0}^n f(i, \bar{x})$ are real recursive.

Now we use another notion of the classical theory of computability in the analog context. A set $S \subset R$ is called a real recursive set if it has a real recursive characteristic function.

Proposition 2.6. *The sets of integer numbers* Z, *natural numbers* N, *rational numbers* Q, *the set of all algebraic reals are real recursive sets.*

Proof. For Z it is sufficient to use $\delta(sin\pi x)$ as a characteristic function χ_Z. Then $\chi_N(x) = \chi_Z(x)\Theta(x)$.

For Q a construction is more troublesome. Let us start with an auxiliary function $f(n,x) = \prod_{i=1}^{n}(1-\chi_Z(xi))$. Such a function is equal to 0 iff x is of the form $\frac{p}{q}, p, q \in Z$, where $1 \leq q \leq n$, otherwise 1. Going to infinity we can define

$$\chi_Q(x) = 1 - \lim_{z\to\infty} f(\lfloor z \rfloor, x).$$

Let us use a letter A as a symbol of the set of algebraic number. If $x \in A$, then there exists such a polynomial P of some degree n with natural coefficients $a_0, \ldots, a_n$ and vector of natural numbers $i_0, \ldots, i_n$ (describing signs of the coefficients of P) such that $\sum_{j=0}^{n} a_j(-1)^{i_j}x^j = 0$. We would like encode these two vectors into two natural numbers a, i. The known result (see [12]) says us that there exists such a (natural) primitive recursive function β that $\beta(a,0) = n, \beta(a,j) = a_{j-1}, 1 \leq j \leq n+1$, $\beta(i,0) = n, \beta(i,j) = i_{j-1}, 1 \leq j \leq n+1$. Then the value of P for x is given as $P(x,a,i) = \sum_{j=0}^{\beta(a,0)} \beta(a,j-1)(-1)^{\beta(i,j-1)}x^j$. Because natural primitive recursive functions are real recursive (see [5]), hence P is a real recursive function. Let us extended this function into
$P'(x,y,z) = \begin{cases} 1 & y \text{ or } z \text{ are not natural numbers or } \beta(y,0) \neq \beta(z,0), \\ P(x,y,z) & \text{otherwise.} \end{cases}$

Then x is an algebraic number only in this case iff there exist such y, z that $P'(x,y,z) = 0$. This condition can be checked by the following construction: let $p(x,y,w) = |P'(x,y,w)| + 1$ for $P'(x,y,z) \neq 0$, otherwise 0;
then $p'(x,z) = \prod_{k=0}^{\lfloor z \rfloor}\prod_{j=0}^{\lfloor z \rfloor} p(x,k,j)$ will be equal to zero only in this case if there exist such a polynomial P encoded by $k, j \leq \lfloor z \rfloor$, that its value in x is equal to 0. Now to find a characteristic function of A suffices to write:

$$\chi_A(x) = \begin{cases} 1 \; \eta_z p'(x,z) = 1 \text{ and } \lim_{z\to\infty} p'(x,z) = 0, \\ 0 \text{ otherwise.} \end{cases}$$

□

3 η-Hierarchy

Here we aproach a new problem. Are there different levels of difficulty in a computation if it goes beyond the Turing computability? The natural measure of a function's difficulty can be join with the degree of (dis)continuity. The above considerations lead us to the conception of η-hierarchy which describe the level of nesting limits in the definition of a given function.

We should start with the notion of syntactic n-ary descriptions of real recursive vectors. Let us introduce some kind of symbols called basics descriptors for all basic real recursive functions. The combination of such descriptions for given real recursive functions will form a new description of another function. Let us start with basic functions: i_k^j is a k-ary description for projection I_k^j for all $1 \leq j \leq k$; $1_k, \overline{1}_k, 0_k$ are k-ary descriptions for constants $1, -1, 0$ used with k variables. We must add also operator symbols (descriptors) for all introduced operators: dr - for a differential recursion, c - for a composition, l, ls, li for a respective kind of limits (lim, lim sup, lim inf). The collection of descriptors of real recursive vectors can be inductively defined as follows: $i_n^j, 1_n, \bar{1}_n, 0_n$ are n-ary descriptions of I_n^j, $1 \leq j \leq n \in N$, $f(x_1, \ldots, x_n) = 1$, $f(x_1, \ldots, x_n) = -1$,$f(x_1, \ldots, x_n) = 0$ for all $(x_1, \ldots, x_n) \in R^n$, $n \in N$, respectively. If $\langle h \rangle = \langle h_1, \ldots, h_m \rangle$ is a k-ary description of the real recursive vector h and $\langle g \rangle = \langle g_1, \ldots, g_k \rangle$ is a n-ary description of the real recursive vector g, then $c(\langle h \rangle, \langle g \rangle)$ is a n-ary description of the composition of h and g. For differential recursion we can write: if $\langle h \rangle = \langle h_1, \ldots, h_n \rangle$ is a k-ary description of the real recursive vector h and $\langle g \rangle = \langle g_1, \ldots, g_n \rangle$ is a $k + n + 1$-ary description of the real recursive vector g, then $dr(\langle h \rangle, \langle g \rangle)$ is a $k+1$-ary description of the solution of the Cauchy problem for h, g (if such a solution exists). Finally, if $\langle h \rangle = \langle h_1, \ldots, h_m \rangle$ is a $n + 1$-ary description of the real recursive vector h, then $l(\langle h \rangle)$, $li(\langle h \rangle)$, $ls(\langle h \rangle)$ is a n-ary description of an apropriate infinite limit (respectively $\lim, \liminf, \limsup$) of h (if such limits exist).

Definition 3.1. *For a given n-ary description s of a vector f let $E_i^k(s)$ (the η-number with respect to i-th variable of the k-component) be defined as follows: $E_i^1(0_n) = E_i^1(1_n) = E_i^1(\bar{1}_n) = 0$; $E_i^m(c(\langle h \rangle, \langle g \rangle)) = \max_{1 \leq j \leq k}(E_j^m(\langle h \rangle) + E_i^j(\langle g_j \rangle))$, where h is a n components k-ary vector and g is a k-components m-ary vector. For a differential recursion we distinguish two cases: if $i \leq k$ then $E_i^j(dr(\langle f \rangle, \langle g \rangle)) =$
$\max(E_i^1(\langle f_1 \rangle) \ldots, E_i^1(\langle f_n \rangle), E_i^1(\langle g_1 \rangle) \ldots, E_i^1(\langle g_n \rangle), E_{k+1}^1(\langle g_1 \rangle), \ldots, E_{k+1}^1(\langle g_n \rangle))$; otherwise if $i = k + 1$: $E_i^j(dr(\langle f \rangle, \langle g \rangle)) =$
$\max_{0 \leq m \leq n}(\max(E_{k+m+1}^1(\langle g_1 \rangle), \ldots, E_{k+m+1}^1(\langle g_n \rangle)))$ where f is a n components k-ary vector and g is a n components $k + n + 1$-ary vector. Finally for limits we have $E_i^k(l(\langle h \rangle)) = E_i^k(li(\langle h \rangle)) = E_i^k(ls(\langle h \rangle)) = \max(E_i^k(\langle h \rangle), E_{n+1}^k(\langle h \rangle)) + 1$, where h is a k components $n + 1$-ary vector.*

For the n-ary description s of m components we can define now $E(\langle h \rangle) = \max_k \max_i E_i^k(\langle h \rangle)$ for $1 \leq i \leq n, 1 \leq k \leq m$. Now we can deal with the η-number for a real recursive functions where $\eta(f)$ can be defined as the minimum of $E(\langle f \rangle)$ for all possible descriptions of the function f. We are ready to conclude with definition of η-hierarchy as a family of $H_j = \{f : \eta(f) \leq j\}$.

Let us start with recalling of some real recursive functions from previous propositions.

Example 3.1. From the functions given in Proposition 2.3, we have
$+, \times, -, \exp, \sin, \cos, \lambda x. \frac{1}{x}$, $/$, $\ln, \lambda xy.x^y$ are in H_0, the Kronecker δ function, the signum function and absolute value are in H_1. The Heaviside function Θ, the binary maximum max, the square-wave function and the floor function are in H_1.

To see that our framework is strongly supported by one such classical theory of computation (Shannon's Theory of Analog Computation), we add physical realizability to the basis of recursive functions over the reals stated as the fact that a subclass of H_0 coincides with the functions GPAC-computable. Detailed proof of this statement can be found in [7]. Let us give here the examples of some functions which have the important significance in mathematics and can be expressed in terms of real recursiveness. Let us point out that Rubel showed in [15] incompleteness of the GPAC-computable functions proving the Euler's Γ-function and the Riemann ζ-function are not GPAC-computable.

Example 3.2. The Euler's Γ-function is real recursive function from the class H_1.

Let us recall that Laplace transform of $t^x, x > -1$, is equal to $\frac{\Gamma(x+1)}{s^{t+1}}$, hence $\Gamma(y)$ for $y > 0$ is real recursive and (because Laplace transform uses only one limit) in H_1. Let us add that Marion Pour-El (see [14]) proved that Γ is not GPAC-computable so its class is most probably strictly H_1. By simple construction we give the same result for the Riemann ζ-function. We can also add the corollaries of the constructions used in Propositions 2.5, 2.6.

For given function f from the class $H_i, i \geq 0$, its iteration $F(n,x) = f^n(x)$ is in the class $H_{\max(1,i)}$.

By the class of some set we understand the class of its characteristic function. Then the sets of natural and integer numbers are in H_1, the set of rational numbers Q is in H_3, and the set of algebraic numbers is in H_5.

4 The Halting Problem

Now we can turn to some application of the η operator. We consider a possibility of a process of Turing machines simulation by real recursive functions. Such problems were considered by Moore [9], however his assumptions were connected with a wrong established η-operator. A Turing machine is here understood as usual, more complete description can be found in [11], where is also a proof of the below proposition.

Proposition 4.1. *There are real recursive functions from the class H_1, which can simulate any Turing machine.*

Let us signal a few important questions concerned to Turing machines. The first problem is known as the halting problem: does the machine M for some input reach the final state? There is not a natural recursive characteristic function of this problem. The method of simulation of Turing machines given above can resolve it in the simple way with real recursive functions.

Proposition 4.2. *For any Turing machine M, there exists a real recursive function the class H_3 which is the characteristic function of the halting problem for M.*

The proof given in [11] uses a construction of a sequence of configurations. To check whether this sequence is ended by a configuration with some final step or it is infinite the η-operator is taken.

5 Arithmetical Hierarchy and Computable Numbers

We will proceed now with the relations of natural numbers taken from the arithmetical hierarchy. The class $\Sigma_0^0 = \Pi_0^0$ contains only such relations, which have recursive characteristic functions. The upper stages of this hierarchy can be constructed from the lower ones in the following way:$\Sigma_{n+1}^0 = \{P : (\exists P' \in \Pi_n^0)P(\bar{m}) \equiv \exists s P'(\bar{m}, s)\}$, $\Pi_{n+1}^0 = \{P : (\exists P' \in \Sigma_n^0)P(\bar{m}) \equiv \forall s P'(\bar{m}, s)\}$, where $P \subseteq N^k, P' \subseteq N^{k+1}, k \geq 1$. To complete our hierarchies we can add the following equation $\Delta_n^0 = \Sigma_n^0 \cap \Pi_n^0, n \geq 0$. Now let us correlate this infinite hierarchy of sets and relations to the η-hierarchy. We must return to the Turing machine and its simulation by real recursive functions.

From Proposition 4.1 and from the fact that all natural recursive sets and relations have Turing computable total characteristics we get the following conclusion:

Proposition 5.1. *Every natural recursive set or relation is in H_2, i.e. $\Sigma_0^0 = \Pi_0^0 \subset H_2$.*

The next element of our investigation has to deal with higher levels of arithmetical hierarchy. For this purpose we need to analyse the method of use of quantifiers. For every function $f : R^{n+1} \to R$ we can construct such real recursive function $\rho_f : R^n \to R$ that

$$\rho_f(\bar{x}) = \begin{cases} 1 \ \exists y \in N f(\bar{x}, y) = 0, \\ 0 \ \forall y \in N f(\bar{x}, y) \neq 0. \end{cases}$$

To this effect we start with a description of the function $f_c(\bar{x}, y) = 1 - \delta(f(\bar{x}, y))$. This function has the following property $f_c(\bar{x}, y) = 1 \equiv f(\bar{x}, y) \neq 0$, $f_c(\bar{x}, y) = 0 \equiv f(\bar{x}, y) = 0$. It is easy to observe that now

$$\lim_{z \to \infty} \prod_{j=0}^{z} f_c(\bar{x}, j) = \begin{cases} 0 \ \exists y \in N f(\bar{x}, y) = 0, \\ 1 \ \forall y \in N f(\bar{x}, y) \neq 0. \end{cases}$$

Hence $\rho_f(\bar{x}) = 1 - \lim_{z \to \infty} \prod_{j=0}^{\lfloor z \rfloor} f_c(\bar{x}, j)$. Finally, by properties of the iteration, we can claim that $\rho_f \in H_{i+2}$. From the above considerations we can deduce the following theorem.

Proposition 5.2. *The sets and relations from Σ_i^0, Π_i^0 belong to H_{i+2} for $i \geq 0$.*

Let us add that by computable reals (points) we understand values of real recursive functions with an arity 0.

We can prove that all real numbers given by a continued fraction built from real recursive sequences of naturals are real recursive, and conversely that for a real number its continued fraction expansion can be described by a real recursive function. Continued fractions, together with the search for zeroes' operator, can be used to implement the Analytic Hierarchy in the same way as Cris Moore did it in [9].

Proposition 5.3. *Let x be a real number given as a continued fraction $x = [x_0, x_1, x_2, \ldots]$:*

$$x = x_0 + \cfrac{1}{x_1 + \cfrac{1}{\ddots}}.$$

Then $\phi(x, n) = x_n$ is a real recursive function. Conversely if $f : R \to R$ is a real recursive function, which maps natural numbers to natural numbers, then $x[f] = [f(0), f(1), f(2), \ldots]$ is a real recursive number.

Proof. For the first part of this proposition it is sufficient to define[3] $\phi(x, n) = \lfloor g^n(x) \rfloor$, where $g(x) = \begin{cases} 0 & x \in Z, \\ \frac{1}{x - \lfloor x \rfloor} & x \notin Z. \end{cases}$

Conversely, for a given f we use the real recursive map

$$t(x, k) = \begin{cases} (\frac{1}{x} + f(k-1), k-1), & k > 0, \\ (x, 0), & k = 0. \end{cases}$$

Now if $T(k) = I_2^1(t^k(f(k), k))$, then

$$T(k) = f(0) + \cfrac{1}{f(1) + \cfrac{1}{\ddots + \frac{1}{f(k)}}}$$

and we can find x as $\lim_{y \to \infty} T(\lfloor y \rfloor)$. □

In this sense e, π are computable reals: $\pi = 3 + [7, 15, 1, 292, 1, \ldots], e = 2 + [1, 2, 1, 1, 4, 1, 1, \ldots, 2n, 1, 1, \ldots]$. Let the continued fraction for x be written $[x_0, x_1, \ldots]$. Then the limiting value of the geometric mean is almost always Khinchin's constant (failing only for a countable number of reals) $K = \lim_{n \to \infty} \sqrt[n]{x_0 \ldots x_n}$, which is real recursive number as: $K = \lim_{n \to \infty} \prod_{k=1}^{n} (1 + \frac{1}{k(k+2)})^{\frac{\ln k}{\ln 2}}$.

We can also prove that many real numbers which are not computable in Turing sense are real recursive. Let us choose some function $f : N \to \{0, 1\}$ with a graph $G_f(x, y) \equiv y = f(z)$ which belongs to the class $\Delta_j, j > 1$. Then we can construct the number x equal to $\lim_{n \to \infty} \sum_{i=0}^{n} (\frac{3f(i)}{4^i} + \frac{1 - f(i)}{4^i})$, which is uncomputable in Turing sense. However in the obvious manner it is real recursive.

Finally let us mention a real recursive character of a particular Turing uncomputable number, namely Chaitin's constant.

Proposition 5.4. *Chaitin's Ω constant is a real recursive number.*

Proof. As usual let $\Omega = \sum_p 2^{-|p|}$, where p is a binary representation of halting programs (without inputs) of Turing machine with a property, that no proper prefix of a syntactically correct program is a syntactically correct program.

Let U be some Turing universal machine working on Turing programs without inputs given on a tape by a specific binary coding. This coding has such a

[3] In the case $n = 0$ we have $g^0(x) = x$.

property that if a binary sequence w encodes a syntactically correct program, then no proper prefix of w encodes a syntactically correct program. It can be simply obtain, for example by a convention that the beginning of w contains a length of w. Let us assume that $b_n(i)$ is a binary representation of natural number i given by n digits (possibly with zeroes at the beginning).

We can define

$$F(n) = \sum_{i=1}^{n} [\sum_{j=0}^{2^{i+1}-1} H'_U(0, b_{i+1}(j))] 2^{-(i+1)},$$

where H'_U is a real recursive function with such a property that $H'(0, x) = 0$ iff x does not encode a syntactically correct program, or if x encodes a correct program which is not halting; otherwise H' has the value 1. This function can be easily obtained from a characteristic function H_U of the halting problem for U, because checking a syntactical correctness of a program can be done by a natural total recursive function (hence by a real recursive function too). An existance of such a function is guaranteed by Proposition 4.2. A tape is fulfilled only by a binary sequence $b_{i+1}(j)$ with a length $i + 1$. The final step is given as $\Omega = \lim_{n\to\infty} F(n)$. $\square$

6 Conclusions

We introduced a framework of real recursive functions (i. e. an inductive set) in such a way that (a countable set of) functions over the reals exist that simulate arbitrary Turing machines, decide the halting problem, and decide all levels of the arithmetic hierarchy. Such a class of functions includes in a very natural way the elementary functions of Analysis, and real numbers computable in the Turing sense. The main ingredients with regard to Kleene's theory, are the following closure operators: a scheme of differential recursion substitutes for recursion and the taking of infinite limits substitutes for minimalization.

Assume for simplicity that with limits we can decide whenever a real-valued function is in C^0 for non negative values[4]. We know that the classical problem of knowing if a given unary computable function is everywhere 0 is undecidable: this undecidability result is based on standard methods like reducibility via s-m-n theorem. With the toolbox of Analysis we have different but nevertheless standard methods too: We can take the absolute value of a given such function f, namely $|f(x)|$, and integrate from 0 to infinity; we then have $I(f) = \int_0^\infty |f(x)| dx = 0$ if and only if f is 0 in $[0, \infty)$; $\delta(I(f))$ provides a characteristic to such a problem. We believe that our most general framework, envolving infinite limits, have enough ingredients to allow the translation of classical computability and classical computational complexity problems into Analysis. We do believe that such translations might be a solution to open problems described in analytic terms: we are now much envolved in the definition of analog classes P and NP.

[4] We are ready to give details of such method in the forthcoming paper.

Acknowledgements We would like to thank Cris Moore for many discussions on the more obscure results presented in his seminal paper [9]. Our paper was design to present a most general framework for the Recursion Theory on the Reals, showing that Moore's original ideas can still be fully implemented in our conceptual scheme.

References

1. A. Ben-Hur, H.T. Siegelmann and S. Fishman. A theory of complexity for continuous time systems. *Journal of Complexity*, 18(1): 87-103, 2002.
2. L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*, Springer, 1998.
3. M. S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, 138(1):67-100, 1995.
4. M. L. Campagnolo. Computational complexity of real valued recursive functions and analog circuits, PhD dissertation, Universidade Tecnica de Lisboa, 2001.
5. M. L. Campagnolo, C. Moore, and J. F. Costa. Iteration, inequalities, and differentiability in analog computers. *Journal of Complexity*, 16(4):642-660, 2000.
6. M. L. Campagnolo, C. Moore, and J. F. Costa. An analog characterization of the Grzegorczyk hierarchy. *Journal of Complexity*, 18(4):977–1000, 2002.
7. D. Graça and J. F. Costa. Analog computers and recursive functions over the reals. *Journal of Complexity*, 19(5): 644-664, 2003.
8. W. Thomson (Lord Kelvin). On an instrument for calculating the integral of the product of two given functions. *Proc. Royal Society of London*, 24: 266-268, 1876.
9. C. Moore. Recursion theory on the reals and continuous-time computation. *Theoretical Computer Science,* 162:23-44, 1996.
10. J. Mycka. μ-Recursion and infinite limits. *Theoretical Computer Science,* 302:123-133, 2003.
11. J. Mycka and J. F. Costa. Real recursive functions and their hierarchy, Journal of Complexity, 20(6): 835-857, 2004.
12. P. Odifreddi. *Classical Recursion Theory*, North-Holland, 1989.
13. P. Orponen. A survey of continuous-time computation theory. In D.-Z. Du and K.-I. Ko, (eds), *Advances in Algorithms, Languages and Complexity*, 209-224, Kluwer Academic Publishers, 1997.
14. M. B. Pour-El. Abstract computability and its relations to the general purpose analog computer. *Transactions Amer. Math. Soc.*, 199:1-28, 1974.
15. L. A. Rubel. Some mathematical limitations of the general-purpose analog computer. *Advances in Applied Mathematics,* 9:22-34, 1988.
16. L. A. Rubel. The extended analog computer. *Advances in Applied Mathematics*, 14:39-50, 1993.
17. C. Shannon. Mathematical theory of the differential analyzer. *J. Math. Phys. MIT*, 20:337-354, 1941.
18. H. T. Siegelmann and S. Fishman. Analog computation with dynamical systems. *Physica D*, 120: 214-235, 1998.
19. J. Traub and A. G. Werschulz. *Complexity and Information*, Cambridge University Press, 1998.
20. A. Vretblad. *Fourier Analysis and Its Applications*, Springer-Verlag, 2003.

Abstract Geometrical Computation for Black Hole Computation
(Extended Abstract)

Jérôme Durand-Lose[⋆]

Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans,
B.P. 6759, F-45067 ORLÉANS Cedex 2.

Abstract. The Black hole model of computation provides super-Turing computing power since it offers the possibility to decide in finite (observer's) time any recursively enumerable ($\mathcal{R}.\mathcal{E}.$) problem. In this paper, we provide a geometric model of computation, *conservative abstract geometrical computation*, that, although being based on rational numbers, has the same property: it can simulate any Turing machine and can decide any $\mathcal{R}.\mathcal{E}.$ problem through the creation of an accumulation. Finitely many signals can leave any accumulation, and it can be known whether anything leaves. This corresponds to a black hole effect.

Key-words: Abstract geometrical computation, Black hole model, Energy conservation, Malament-Hogarth space-time, Super-Turing computation, Turing universality, Zeno phenomena.

None of the physicist aspects of this paper is to be considered as definitely true. The author, being a computer scientist with little knowledge on the matter, would not feel insulted if one would consider these mere inventions/illusions. However, we do not pretend to explain or describe black holes, but just to provide a computer scientist insight and model mostly directed to the computer science community. This paper could have been presented as a model of computation with special features, but since so much similarities exist, we stress on the correspondence with the Black hole model.

1 Introduction

Theoretical physicists address the limits of the Church-Turing thesis as they get insights of possible space-times abiding Einstein's equations but providing super-Turing computing power [Hog94]. The idea is to have the possibility to use an infinite amount of time on a separate future endless curve to try solving a recursively enumerable ($\mathcal{R}.\mathcal{E}.$) problem, such that the result, or the absence of any result, can be retrieved in finite time in the main curve. For the theoretical computer scientist, this is related to infinite Turing computation or computation on ordinals [Ham02].

[⋆] This research was done while the author was member of LIP, ÉNS Lyon and of Université de Nice-Sophia Antipolis, France.

Malament-Hogarth space-times [Hog00, EN02] provides this. Roughly speaking, the idea is the following. General relativity permits space-times in which time runs with different "speeds" in different regions. We arrange the life-lines of the computer and the observer in such a way that the machine has an infinite amount of time ahead of it; but any signal it returns is received by the observer within a bounded local delay (measured on the observer's clock). After this finite delay, the observer knows whether the computation ever stopped (by noticing whether anything was received) and what the answer is. It is thus possible to decide any $\mathcal{R}.\mathcal{E}.$ problem in finite time.

Abstract geometrical computation [DL04] considers Euclidean lines. The support of space and time is thus $\mathbb{R}$. Computations are produced by *signal machines* which are defined by a finite set of *meta-signals* and a finite set of *collision rules*. Signals are atomic information, corresponding to meta-signals, moving at constant speed thus generating Euclidean line segments on space-time diagrams. Collision rules are pairs *(incoming meta-signals, outgoing meta-signals)*, that define a mapping (which means determinism) over sets of meta-signals. They define what happens when signals meet, *i.e.* at the extremities of the line segments.

A configuration (at a given time or the restriction of the space-time diagram to a given time) is a mapping from $\mathbb{R}$ to meta-signals, collision rules, and two special values: void (*i.e.* nothing there) and accumulations (amounting for black holes). There should be finitely many positions not mapped to void. The time scale is $\mathbb{R}^+$, so that there is no such thing as a "next configuration". The following configurations are defined by the uniform movement of each signal, the speed of which is defined by its associated meta-signal. When two or more signals meet, this produces a *collision* defined by a collision rule. In the configurations following a collision, incoming signals are removed and outgoing signals corresponding to the outgoing meta-signals are added.

Zeno like acceleration and accumulation can be constructed as on the right of Fig. 1. This provides the black hole-like artifact for deciding $\mathcal{R}.\mathcal{E}.$ problems. But accumulations can lead to an uncontrolled burst of signals producing infinitely many signals in finite time (as in the right of Fig. 1). In order to avoid this, we impose a *conservativeness* condition on the rules: a positive energy is defined for every meta-signal, the sum of these energies must be conserved by each rule. Thus no energy creation is possible, and the number of signals is bounded.

Each signal corresponds to a meta-signal which indicates its slope on the space-time diagram. Since there are finitely many meta-signals, there are finitely many slopes. This limitation may seem restrictive and unrealistic, even awkward as a quantification inside an analog model of computation. Let us notice that, first, it comes from cellular automata (CA) (as explained below): once a discrete line is identified, wherever (and whenever) the same pattern appears, the same line is expected, thus with the same slope. Second, we give two pragmatic arguments: (1) laws to compute new slopes in collisions are not so easy to design and pretty cumbersome to manipulate; (2) there is already much computing power.

Abstract geometrical computation comes from the common use in literature of Euclidean lines to model discrete lines in the space-time diagram of CA to access dynamics or to design. Cellular automata form a well known and studied model of computation and simulation. Configurations are $\mathbb{Z}$-arrays of cells the states of which belong to a finite set. Each cell can only access the states of its neighboring cells. All cells are updated iteratively and simultaneously. The main characteristics of CA, as well as abstract geometrical computation, are: parallelism, synchrony, uniformity and locality of updating. The space-time diagrams of CA are colorings of $\mathbb{Z}\times\mathbb{N}$ with states. Discrete lines are often observed on these diagrams. They can be the keys to understanding the dynamics and correspond to so-called *particles* or *signals* as in, *e.g.*, [Ila01, pp. 87–94] or [BNR91, HSC01]. They can also be the tool to design CA for precise purposes and then named *signals* and used for, *e.g.*, prime number generation [Fis65], firing squad synchronization [VMP70, Maz96] or reversible simulation [DL97]. These discrete line systems have also been studied on their own [MT99, DM02]. All these papers, and many more, implicitly use abstract geometrical computation.

Before presenting our results, we want to convince the reader that it is not just "one more model of computation". First, it does not come "out of the blue" because of its CA origin. Second, to our knowledge[1], it is the only model that is a dynamical system with continuous time and space but finitely many local values. The closest model we know of is the Mondrian automata of Jacopini and Sontacchi [JS90]. Their space-time diagrams are mappings from $\mathbb{R}^n$ to a finite set of colors. They should be bounded finite polyhedra; we are only addressing lines –faces are not considered– and our diagrams may be unbounded and accumulation may happen (they just forbid them). Another close model is the piecewise-constant derivative system [AM95, Bou99]: $\mathbb{R}^n$ is partitioned into finitely many polygonal regions; the trajectory is defined by a constant derivative on each region, thus an orbit is a sequence (possibly over an ordinal) of (Euclidean) line segments. This model is sequential –there is only one "signal"– and the faces that delimit the regions are artifacts that do not exist in our model. Nevertheless, it also uses accumulations to decide $\mathcal{R}.\mathcal{E}.$ problems.

In this paper, space and time are restricted to rationals. This is possible since all the operations used preserve rationality. All intervals should be understood over $\mathbb{Q}$, not $\mathbb{R}$. Extending the definitions to real values is automatic but only the rational case is addressed here.

After formally defining our model in Sect. 2, we rapidly show that any Turing-computation can be carried out through the simulation of 2-counter automata in Sect. 3. The values of the counters are encoded by positions (fixed signals indicates the scale) and the instructions are going forth and back between them. The continuous nature of space is used here: all $1/2^n$ positions exist.

In Sect. 4, we show how to bound temporally a computation that is already spatially bounded. This method is constructive and relies on the continuous nature of space and time. The construction generates an accumulation. We explain how to use these accumulations for deciding $\mathcal{R}.\mathcal{E}.$ problems in Sect. 5. Conclusion, remarks and perspectives are gathered in Sect. 6.

[1] A brief tour of analog/super-Turing models of computation can be found in [DL03, Chap. 2].

2 Definitions

Abstract geometrical computations are defined by the following machines:

Definition 1 A *signal machine* is defined by (M, S, R) where M (*meta-signals*) is a finite set, S (*speeds*) is a mapping from M to $\mathbb{Q}$, and R (*collision rules*) is a subset of $\mathcal{P}(M)\times\mathcal{P}(M)$ that corresponds to a partial mapping of the subsets of M of cardinality at least 2 to the subsets of M (both domain and range are restricted to elements of different speeds).

The elements of M are called *meta-signals*. Each instance of a meta-signal is a *signal* which corresponds to a line segment in the space-time diagram. The mapping S assigns rational *speeds* to meta-signals, *i.e.* the slopes of the segments. The set R defines the *collision rules*, noted $\rho^-\!\rightarrow\!\rho^+$: what happens when two or more signals meet. It also defines the intersections of the segments. The signal machines are deterministic because R must correspond to a mapping.

The *extended value set*, V, is the union M and R plus two symbols: one for void, $\oslash$, and one for an accumulation (or black hole) ❊. A *configuration*, c, is a total mapping from $\mathbb{Q}$ to V such that the set $\{\, x \in \mathbb{Q} \mid c(x) \neq \oslash \,\}$ is finite.

A signal corresponding to a meta-signal μ at a position x, *i.e.* $c(x) = \mu$, is moving uniformly with constant speed $S(\mu)$. A signal must start in the initial configuration or be generated by a collision. It must end in a collision or in the last configuration. This corresponds to condition 2. in Def. 2. At a $\rho^-\!\rightarrow\!\rho^+$ collision, all, and only, signals corresponding to the meta-signals in ρ^- (resp. ρ^+) must end (resp. start). No other signal should be present. This corresponds to 3. in Def. 2. A black hole corresponds to an accumulation of collisions and disappears without a trace. This corresponds to 4. in Def. 2.

Let S_{min} and S_{max} be the minimal and maximal speeds (*i.e.* the extrema of S). The *causal past*, or *light-cone*, arriving at position x and time t, $J^-(x,t)$, is defined by all the positions that might influence through signals, formally:

$$J^-(x,t) = \{\ (x',t') \mid (0 \leq S_{max}(t-t') - x+x') \wedge (0 \leq x-x' - S_{min}(t-t'))\ \}\ .$$

Before formally defining the dynamics by space-time diagrams, we want to point out the black hole formation example of Fig. 1. This example is so simple (*i.e.* 4 meta-signals and 2 collision rules) that such a situation cannot be excluded.

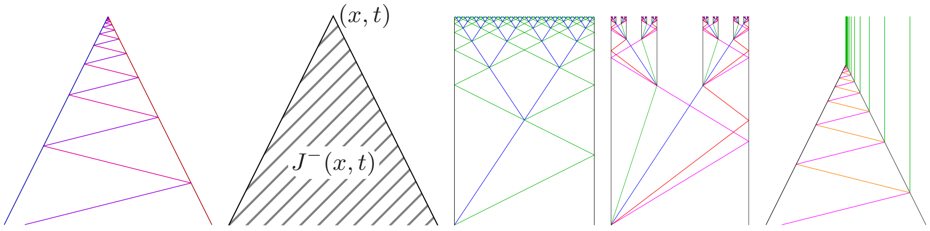


Fig. 1. Black hole, light-cone and unwanted phenomena.

Definition 2 The *space-time diagram*, or orbit, issued from an initial configuration c_0 and lasting for T [2], is a mapping c from $[0,T]$ to configurations (*i.e.* a mapping from $\mathbb{Q} \times [0,T]$ to V) such that, $\forall (x,t) \in \mathbb{Q} \times [0,T]$:

1. $\forall t \in [0,T]$, $\{\, x \in \mathbb{Q} \mid c_t(x) \neq \oslash \,\}$ is finite,
2. if $c_t(x)=\mu$ then $\exists t_i, t_f \in [0,T]$ with $t_i<t<t_f$ or $0=t_i=t<t_f$ or $t_i<t=t_f=T$ s.t.:
   - $\forall t' \in (t_i,t_f)$, $c_{t'}(x+S(\mu)(t'-t)) = \mu$,
   - $t_i = 0$ or $c_{t_i}(x_i) \in R$ and $\mu \in (c_{t_i}(x_i))^+$ where $x_i = x + S(\mu)(t_i - t)$,
   - $t_f = T$ or $c_{t_f}(x_f) \in R$ and $\mu \in (c_{t_f}(x_f))^-$ where $x_f = x + S(\mu)(t_f - t)$;
3. if $c_t(x) = \rho^- \to \rho^+ \in R$ then $\exists \varepsilon$, $0 < \varepsilon$, $\forall t' \in [t-\varepsilon, t+\varepsilon]$, $\forall x' \in [x-\varepsilon, x+\varepsilon]$,
   - $c_{t'}(x') \in \rho^- \cup \rho^+ \cup \{\oslash\}$,
   - $\forall \mu \in M, c_{t'}(x')=\mu \Rightarrow \bigvee \begin{cases} \mu \in \rho^- \text{ and } t' < t \text{ and } x' = x + S(\mu)(t'-t)), \\ \mu \in \rho^+ \text{ and } t < t' \text{ and } x' = x + S(\mu)(t'-t)). \end{cases}$
4. if $c_t(x) = ❊$ then
   - $\exists \varepsilon > 0$, $\forall (x',t') \notin J^-(x,t)$ s.t. $|x-x'|<\varepsilon$ and $|t-t'|<\varepsilon$, $c_{t'}(x) = \oslash$,
   - $\forall \varepsilon > 0$, $\big|\, \{\ (x',t') \in J^-(x,t) \mid t-\varepsilon<t'<t \wedge c_{t'}(x') \in R\ \} \,\big| = \infty$.

On space-time diagrams, the traces of signals represent line segments whose directions are defined by $(S(.),1)$ (1 is the temporal coordinate). Collisions correspond to the extremities of these segments. Examples of space-time diagrams are provided by the various figures. Time is always increasing upwards.

The three right space-time diagrams of Fig. 1 provide examples of possible but unwanted cases. They are not compatible with Def. 2 if times after the accumulation are to be considered. In each case, the number of signals is bursting to infinity and black holes are not isolated. This is unwanted because on the one hand it corresponds to the free apparition of energy, and on the other hand we fell that black holes should be dimensionless points. The two remaining space-time diagrams show even more unwanted cases. We thus introduce the following restriction that prevents such cases and corresponds to the energy conservation.

Definition 3 A signal machine is *conservative* when an atomic positive energy is defined for all meta-signals ($E : M \to \mathbb{N}^*$)[3] such that the total energy of the system is preserved, *i.e.* the sum of all the energy of existing signals is a constant of the system. This is equivalent to accept only rules that preserve this energy, *i.e.* the sum of the energy of incoming meta-signals equals the sum of outgoing ones.

Conservativeness is straightforward if the condition on rules is satisfied. If it is not satisfied, it is very easy to built a configuration such that only this rule is used and then the energy is not preserved.

Property 4 *Given a conservative signal machine and an initial configuration, the number of signal in any following configuration, as well as the number of accumulations, is bounded (by the total energy divided by the least atomic energy).*

[2] This definition can easily be extended to the $T = \infty$ case.
[3] Integer are enough, since there are finitely many meta-signals.

Energy can only be lost in "black hole" formation, *i.e.* accumulation. A sub-case of conservativeness is when all the meta-signals have the same energy and the number of in and out meta-signals are always equal. This is the case in the rest of this paper. We chose to present a more complex notion since it is weaker and better suits the physical notion of the energy conservation.

3 Turing-Computation Capability

We prove the Turing-computation power of our model by simulating any 2-counter automaton (a finite automaton couple with two counters, A and B). The possible actions on any counter are *add/subtract* 1 and *branch if non-zero*. These machines can be described with a six-operations (the three aforementioned ones for each of the two counters) assembly language with branching labels as on the left part of Fig. 6 (see [Min67] for more on 2-counter automata).

The simulation is carried out with both counters encoded by relative positions according to two fixed signals zero and one. These two signals form a scale on the diagram. The counter A (resp. B) is encoded by a single signal a (b) at position $\alpha 2^{-a}$ ($\beta 2^{-b}$) as in Fig. 2. The parameter α and β are rationals such that $1 < \alpha < \beta < 2$; this ensures that the signal a (b) is between zero and one unless its value is zero and in such a case it is on the other side of one. Let us note that the values of α and β prevent the signals from occupying the same place and from being on the scale signals. As can be easily checked on the constructions in the rest of this section, they also prevent that any collision happens on an unconcerned signal.

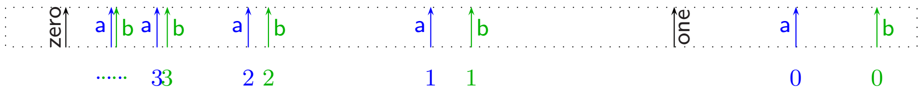


Fig. 2. Encoding positions of counters.

The current instruction (*e.g.* n) is encoded as the signal $\overleftarrow{\mathsf{n}}$. It moves back to zero, bounces, carries out the operation and returns as the next operation. The five possible configurations are given in Fig. 3.

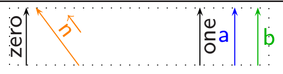
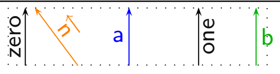
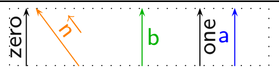
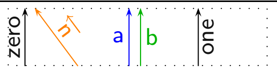
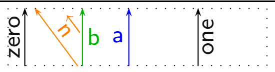
	$a = 0$	$0 < a$
b=0		
$0<b$		 

Fig. 3. Encoding of configurations.

The fact that a signal encoding a counter is on the other side of one only for the value 0 provides an easy way to test whether the counter is zero for branching or subtracting 1: going rightward one is encountered first if and only if the value of the counter is 0.

There are two kinds of meta-signals: 8 for the counters and borders, and the ones generated for the program. The meta-signals of the first kind are: zero, one, a and b of speed 0 used to mark the borders and to encode A and B, and $\overleftarrow{\mathsf{a}}$ ($\overleftarrow{\mathsf{b}}$) and $\overrightarrow{\mathsf{a}}$ ($\overrightarrow{\mathsf{b}}$) of speed -1 and 1 used to increment/decrement A (B). For the second kind, each line n of the program is converted into $\overrightarrow{\mathsf{n}}$ and $\overleftarrow{\mathsf{n}}$ of speed 2 and -2, and possibly $\overrightarrow{\mathsf{n'}}$ and $\overleftarrow{\mathsf{n'}}$ of speed 3 and -3 to carry out increment and decrement as explained below.

First any instruction bounces on zero to be on the left of any other signal and thus be in the right position to start carrying out any instruction. This is achieved by the following rule:

$$\{\mathsf{zero}, \overleftarrow{\mathsf{n}}\} \to \{\mathsf{zero}, \overrightarrow{\mathsf{n}}\}.$$

The full transformation of a program into a signal machine is not given. We only detail the collision rules generated for the most complicated case: a `A--` instruction (at line n). The rules are the following:

$$\{\overrightarrow{\mathsf{n}}, \mathsf{one}\} \to \{\overleftarrow{\mathsf{n{+}1}}, \mathsf{one}\}, \qquad \{\overrightarrow{\mathsf{n}}, \mathsf{a}\} \to \{\overleftarrow{\mathsf{n'}}, \overrightarrow{\mathsf{a}}\},$$
$$\{\mathsf{zero}, \overleftarrow{\mathsf{n'}}\} \to \{\mathsf{zero}, \overrightarrow{\mathsf{n'}}\}, \qquad \{\overrightarrow{\mathsf{n'}}, \overrightarrow{\mathsf{a}}\} \to \{\overleftarrow{\mathsf{n{+}1}}, \mathsf{a}\}.$$

All other rules with $\overrightarrow{\mathsf{n}}$, $\overrightarrow{\mathsf{n'}}$ or $\overleftarrow{\mathsf{n'}}$ are blank, *i.e.*, the same signals are regenerated. The effect of above rules is shown in the space-time diagrams of Fig. 5. The relative position of one and a is very important because a counter already at zero is not decreased. If such is not the case, the distance between zero and a is multiplied by 2 as it can easily be geometrically checked on Fig. 5 where the slopes are indicated by dotted lines.

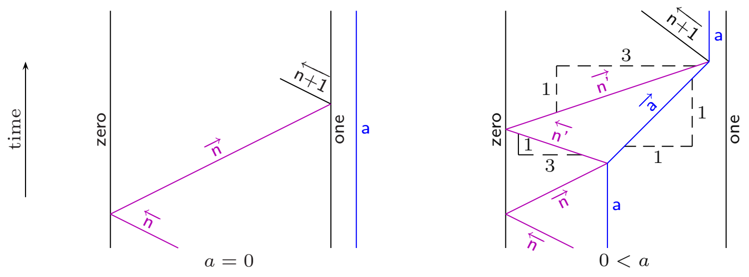


Fig. 4. Implementation of `A--`.

The instructions `A++` does exactly the same thing in reverse, but the zero case does not have to be considered. We do not give the rules, they can be recovered from the left diagram space-time of Fig. 5. The non-zero conditional branching is done by simply noticing that one is met before only if `A` is zero. This is illustrated by the last two space-time diagrams of Fig. 5.

All the instructions on B are carried out similarly.

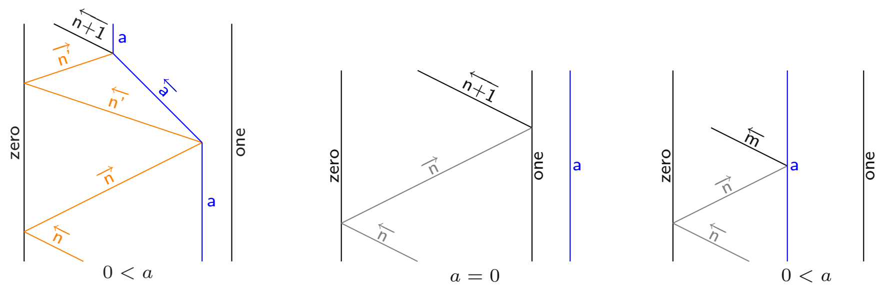


Fig. 5. Implementations of `A++` and n : `IF A!=0` m.

Figure 6 provides three space-time diagrams associated to different initial values. The pictures are strained vertically in order to fit.

```
 beg: B++
      A--
      IF A!=0 beg1
      IF B!=0 imp
beg1: A--
      IF A!=0 beg
 pair: B--
      A++
      IF B!=0 pair
      IF A!=0 beg
 imp: B--
      A++
      A++
      IF B!=0 imp1
      IF A!=0 beg
imp1: B--
      A++
      A++
      A++
      IF B!=0 imp1
      IF A!=0 beg
```

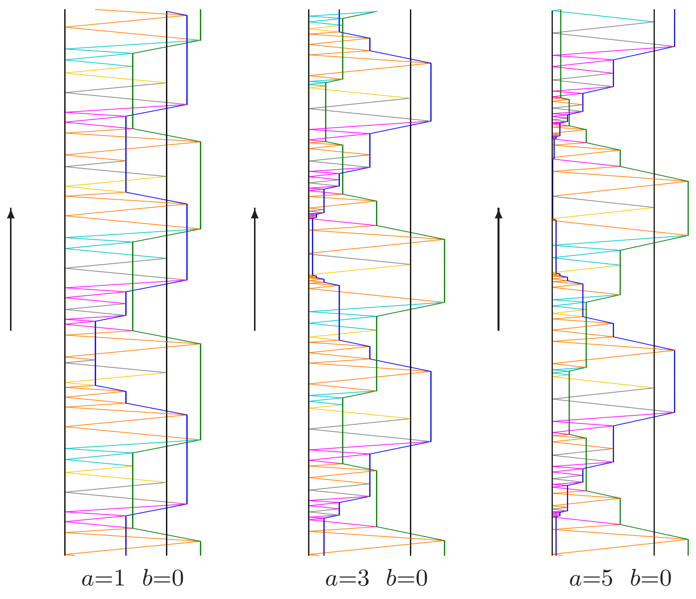


Fig. 6. A 2-counter automaton and its simulations for three different initial values.

The only thing left to consider is the end of the computation, *i.e.* the treatment of the halt. It is not possible to just make the instruction signal disappears since this would yields a non conservative rule. To cope with this, one can choose to let the instruction signal leaves on the left (but this signal could interfere with the rest of the computation), or to let it bounce indefinitely between zero and one; in both cases, the number of signals is preserved.

All together, any 2-counter automaton can be simulated by a conservative signal machine; in fact, any finite number of counters can be included and treated similarly. Signal machines thus form a model of computation which has at least Turing-computing capability.

4 Contraction Principle

It is possible to partially strain any space-time diagram as schematized on Fig. 7. The idea is to decompose the upper part according to two non-collinear vectors. One vector is used as a frontier (here the one of speed β). A change of scale is done on the second one (here multiplication by 3 on the axis corresponding to speed α). This is a strain of a given ratio about the second axe. On Fig. 7, the dotted lines indicate how the images of two points are computed. The grey parts indicate the ongoing computation.

This geometrical transformation is easily implemented inside our model: by switching to strained signal on the frontier, all following computations mimic the unstrained one. The following meta-signals are added: one for the frontier, and one strained meta-signal for each initial meta-signal[4]. All the collision rules are duplicated so that strained signals behave exactly as unstrained ones. Collisions of the form {*frontier and unstrained*}→{*frontier and strained*} are added. New rules are created to account for the possibility of the frontier to pass exactly on a collision. Conservativeness is preserved by setting identical energies to corresponding strained meta-signals.

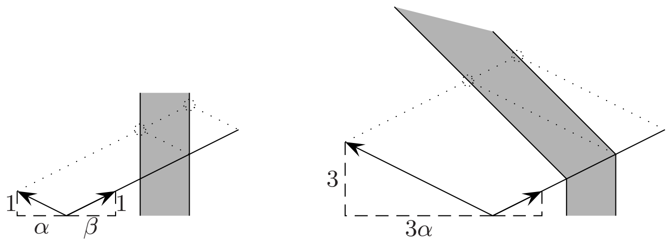


Fig. 7. Strain principle.

With this construction, it is possible to build a structure that scales by one half the rest of the computation as illustrated on Fig. 8. The two directions used correspond to v_0 and $4v_0$, where v_0 is big enough. In the left picture, nothing happens. In the middle picture, the lower signal is the frontier and a strain of ratio 1/2 is done about to the upper signal. In the right picture, a second strain takes place: the role of the directions are exchanged, and the ratio is still 1/2. After the two strains, the computation is scaled by 1/2 on both directions, thus on any direction. The whole computation is scaled by 1/2 and the original meta-signals can be used again since the computation undergoes no strain after the second one. This makes it possible to iterate the shrinking.

[4] Its speed is computed by some $(ax + b)/(cx + d)$ formula whose coefficients depend on the parameters which have to verify some easily satisfied conditions (see [DL03, Chap. 7] for details).

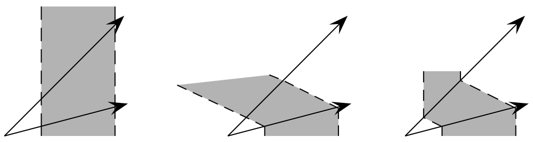


Fig. 8. Shrinking principle.

From now on, only spatially bounded space-time diagrams are considered. This is sufficient to ensure that the computation remains inside the structure when shrinking is iterated. It is possible to add some extra signals to restart the shrinking each time as in the left part of Fig. 9. The right picture represents the application of this structure to a simulation of a 2-counter automaton.

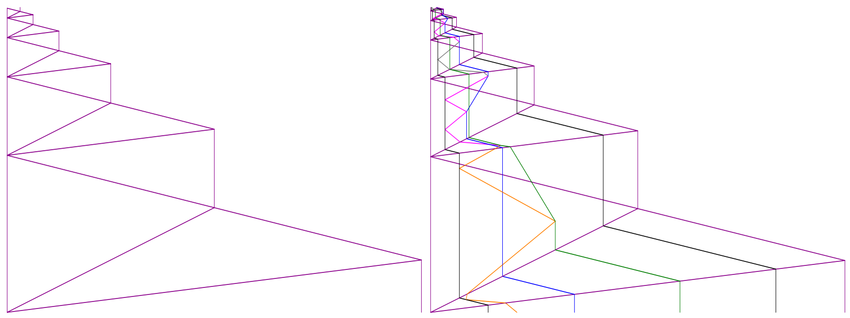


Fig. 9. Iterated shrinking: structure and examples.

In each space-time diagrams of Fig. 9, there is an accumulation point: there are infinitely many collisions accumulating to the upper angle of the triangle. This is a "Zeno effect": finite (continuous) duration but infinitely many (discrete) instants. All collisions are in the light cone ending there (and there is nothing out of it). This corresponds to the accumulation / black hole of Cond. 4 in Def. 2.

5 Black Hole Formation

We consider the simulation of a 2-counter automata such that, when the simulation stops, if the configuration corresponds to acceptance, then the instruction signal goes on the left. In the case of rejection, another signal would be issued.

The iterated shrinking construction is modified in order not to act on this signal (*i.e.* it is always generated unstrained and never strained) so that it leaves the iterated shrinking. The iterated shrinking provides the black hole effect; these specially treated signals represent the information that "leaves" the black hole before the collapsing.

It only remains to get this information or assert that no information had left (*i.e.* the computation never stops). This is done by bounding the iterated shrinking by 2 signals that meet after the black hole. If a notification of acceptance or rejection leaves, they grab it before they meet. So that, at the meeting, they know whether the computation finished. This is illustrated by Fig. 10.

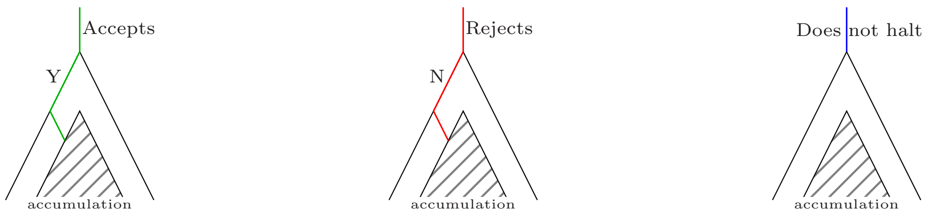


Fig. 10. Encapsulation of a black hole.

6 Conclusion

We provide a geometrical model of computation that is Turing-computation universal and has the special features of the Black hole model. We are not using already existing black holes, but rather creating them on demand (a Malament-Hogarth space-time is implicitly built). It is not so strange that computation forms the black hole since they come from the same matter as the machine sent into. One can also consider that some signals fix black hole formation, while others carry out the computation using the black hole.

One may object that black holes disappearance is not acceptable. The underlying space being one-dimensional, any remaining black hole would form a barrier preventing information to cross from one side to the other; in two and more dimension, it is alway possible to go around it. Another argument is to imagine that signals are drifting in a higher dimensional space, so that the black holes remain, but its orbit is not in the plane of the space-time diagram.

Reversibility is an important issue in theoretical physics. One can easily check that reversibility corresponds to R being one-to-one. At the expense of more complex constructions, universality can be achieved as well as the use of accumulations as black holes. But the final collapsing is not reversible.

The number of possible black holes / $\mathcal{R}.\mathcal{E}.$ queries is bounded from the start (each needs a minimal amount of energy). Unless the black hole returns the energy in some form, which is clearly forbidden here, there is no way to address second order accumulations (*i.e.* ω^2 or second order space-time arithmetic deciding or Σ_2^0 in the arithmetical hierarchy), unless infinitely many signals exist at start, apart one from another. This way it would be reasonably possible to hope to climb the arithmetical hierarchy as in [AM95, Bou99].

As long as the model is restricted to rationals, there are finitely many signals present at any instant and there is no accumulation, the model is Turing-universal and can be simulated by any Turing machine and is thus Turing-equivalent. Real values for speeds and/or positions can be used as oracles and thus provide computing ability that goes beyond Turing-computation.

References

[Ada02] A. Adamatzky, editor. *Collision based computing*. Springer, 2002.

[AM95] E. Asarin and O. Maler. Achilles and the Tortoise climbing up the arithmetical hierarchy. In *FSTTCS '95*, number 1026 in LNCS, pp. 471–483, 1995.

[BNR91] N. Boccara, J. Nasser, and M. Roger. Particle-like structures and interactions in spatio-temporal patterns generated by one-dimensional deterministic cellular automaton rules. *Phys. Rev. A*, 44(2):866–875, 1991.

[Bou99] O. Bournez. Achilles and the Tortoise climbing up the hyper-arithmetical hierarchy. *Theoret. Comp. Sci.*, 210(1):21–71, 1999.

[DL97] J. Durand-Lose. Intrinsic universality of a 1-dimensional reversible cellular automaton. In *STACS '97*, number 1200 in LNCS, pp. 439–450. Springer, 1997.

[DL03] J. Durand-Lose. *Calculer géométriquement sur le plan – machines à signaux*. Habilitation à diriger des recherches, École Doctorale STIC, Université de Nice-Sophia Antipolis, 2003. In French.

[DL04] J. Durand-Lose. Abstract geometrical computation: Turing-computing ability and unpredictable accumulations (extended abstract). Technical Report 2004–09, LIP, ÉNS Lyon, 46 allée d'Italie, 69 364 Lyon 7, 2004.

[DM02] M. Delorme and J. Mazoyer. Signals on cellular automata. in [Ada02], pp. 234–275, 2002.

[EN02] G. Etesi and I. Nemeti. Non-Turing computations via Malament-Hogarth space-times. *Int. J. Theor. Phys.*, 41(2):341–370, 2002. gr-qc/0104023.

[Fis65] P. C. Fischer. Generation of primes by a one-dimensional real-time iterative array. *J. ACM*, 12(3):388–394, 1965.

[Ham02] J. D. Hamkins. Infinite time Turing machines: Supertask computation. *Minds and Machines*, 12(4):521–539, 2002. math.LO/0212047.

[Hog94] M. Hogarth. Non-Turing computers and non-Turing computability. In *Biennial Meeting of the Philosophy of Science Association*, number 1, pp. 126–138, 1994.

[Hog00] M. Hogarth. *Predictability, computability and space-time*. PhD thesis, University of Cambridge, UK, 2000. ftp://ftp.math-inst.hu/pub/algebraic-logic/Hogarththesis.ps.gz.

[HSC01] W. Hordijk, C. R. Shalizi, and J. P. Crutchfield. An upper bound on the products of particle interactions in cellular automata. *Phys. D*, 154:240–258, 2001.

[Ila01] A. Ilachinski. *Cellular Automata –A Discrete Universe–*. World Scientific, 2001.

[JS90] G. Jacopini and G. Sontacchi. Reversible parallel computation: an evolving space-model. *Theoret. Comp. Sci.*, 73(1):1–46, 1990.

[Maz96] J. Mazoyer. On optimal solutions to the Firing squad synchronization problem. *Theoret. Comp. Sci.*, 168(2):367–404, 1996.

[Min67] M. Minsky. *Finite and Infinite Machines*. Prentice Hall, 1967.

[MT99] J. Mazoyer and V. Terrier. Signals in one-dimensional cellular automata. *Theoret. Comp. Sci.*, 217(1):53–80, 1999.

[VMP70] V. I. Varshavsky, V. B. Marakhovsky, and V. A. Peschansky. Synchronization of interacting automata. *Math. System Theory*, 4(3):212–230, 1970.

Is Bosco's Rule Universal?

Kellie Michele Evans

California State University, Northridge, CA 91330-8313, USA
kellie.m.evans@csun.edu,
WWW companion page to paper:
http://www.csun.edu/∼kme52026/bosco/bosco.html

Abstract. The *Game of Life* (Life) is a two-state, two-dimensional, nearest neighbor cellular automaton (CA), which John Horton Conway proved is universal. The *Larger than Life* (LtL) family of CAs generalizes Life to large neighborhoods and general birth and survival thresholds. A specific *threshold-range scaling* of Life to LtL yields *Bosco's rule*, which is a *range 5* CA with dynamics similar to Life. In the 1990s Conway challenged us to prove that rules such as Bosco's are, like Life, universal. Here we show that Bosco's rule supports patterns such as those used in the proof that Life is universal. Specifically, we build a *sliding block memory*, similar to the auxiliary storage device Conway described and claimed could be built. Our construction is based on Life's sliding block memory designed and built by Dean Hickerson in 1990. In a companion paper we explore various questions which have arisen since Conway posed his challenge, including whether the details given in his proof that Life is universal are sufficient and what necessary and sufficient conditions are required to prove that Bosco's rule, or any two-dimensional CA, is universal.

Keywords: *bugs, cellular automata, gliders, Game of Life, Larger than Life, register, sliding block memory, spaceships, universal*

1 Introduction

As is well known, in the late 1960s John Horton Conway wanted to find a cellular automaton (CA) with a simple update rule capable of *universal computation.* His quest to find such a rule was successful and the rule, which he named the *Game of Life* (Life) was so intriguing it generated international interest in CAs [1]. New Life patterns continue today to be discovered and posted online and new Life questions and results emerge regularly [2].

To prove that Life is universal, Conway showed how to define Life patterns that can imitate computers. That is, he showed that Life's patterns can be configured spatially to create *glider guns* as well as AND, OR, and NOT *logical gates*. He also described an *auxiliary storage device*, capable of holding arbitrarily large numbers [3]. His design will be discussed further in Section 3.

In the early 1990s, David Griffeath generalized Life to *Larger than Life* (LtL), which is a family of CAs with large neighborhoods and general birth and survival thresholds [4]. Studying LtL has led to numerous results and open questions about large-range cellular automata [5], [6]. In particular, a specific *threshold-range scaling* of Life to LtL yields a family of "Life-like" rules for each *range*, or neighborhood size. Our favorite such example is *Bosco's rule*, which is a *range 5* CA (meaning the rule's neighborhood is an 11×11 box).

In 1994 we showed several of these Life-like LtL rules to Conway and he challenged us to prove that some of them (which he called "interval rules") are universal. In this paper we show that Bosco's rule supports patterns such as those Conway used to prove that Life is universal. Specifically, we build a *sliding block memory*, similar to the auxiliary storage device he described and claimed could be built. Our construction is based on Life's sliding block memory designed and built by Dean Hickerson in 1990 [7].

The constructions we present in this paper would have been nearly impossible to build when Conway posed his challenge due to the large sizes of the lattices required and the consequential slowness of running such systems on a standard computer. (We had access to a Cellular Automata Machine (CAM), which was lightning fast for its day, however, creating detailed initial states on the CAM was not efficient.) This work is now possible because clock speeds on personal computers have increased dramatically and *Mirek's Celebration* (or MCell), a CA modeling environment, which allows one to create detailed initial states on the fly, has come into existence (MCell is freeware available for download at [8]). In addition, this project needed input from Hickerson, the Life expert mentioned above.

2 Bosco's Rule: Definitions and Notation

Let us define Bosco's rule. Each site of the *two-dimensional lattice* $\mathbf{Z}^2$ is in one of two *states*, *live* (1) or *dead* (0). This is the *initial configuration* of the system. The *neighborhood* $\mathcal{N}_5$ of a site consists of the 11×11 sites in the box surrounding and including it. That is, the neighborhood of the origin is $\mathcal{N}_5 = \{y \in \mathbf{Z}^2 : ||y||_\infty \leq 5\}$, so that its translate $\mathcal{N}_5^x = x + \mathcal{N}_5$ is the neighborhood of site $x \in \mathbf{Z}^2$. $\mathcal{N}_5$ is called the *range* 5 *box neighborhood*. Each *time step*, all of the sites *update* (meaning change states or not) simultaneously according to the deterministic LtL *rule*, which in words is:

• **Birth**: A site that is dead at time t will become live at time $t+1$ if and only if the number of live sites in its neighborhood at time t is in the closed interval $[34, 45]$.

• **Survival**: A site that is live at time t will remain live at time $t+1$ if and only if the number of live sites in its neighborhood (itself included) at time t is in the closed interval $[34, 58]$.

• **Death**: In all other cases a site remains or becomes dead.

Let us introduce the notation needed for the definitions that follow.

Let $\mathcal{T}$ denote Bosco's rule. That is, $\mathcal{T} : \{0,1\}^{\mathbf{Z}^2} \longmapsto \{0,1\}^{\mathbf{Z}^2}$. Let $\xi_t(x) \in \{0,1\}$ denote the state of the site $x \in \mathbf{Z}^2$ at time t and let ξ_t represent the state of all sites in $\mathbf{Z}^2$ at time t. As is customary we will often think of the CA as a set-valued process, confounding ξ_t with $\{x : \xi_t(x) = 1\}$. For example, this allows us to use the notation $\xi_t^{\Lambda} = \mathcal{T}^t(\Lambda)$ to mean that starting from configuration $\xi_0 = \Lambda$ we arrive at the set $\mathcal{T}^t(\Lambda)$ of occupied sites after t iterations of rule $\mathcal{T}$.

Bosco's rule is just one of the numerous range 5 LtL CA rules, which form a four-parameter family indexed by the endpoints β_1 and β_2 of the *birth intervals* and the endpoints δ_1 and δ_2 of the *survival intervals* and denoted by the 5-tuple $(5, \beta_1, \beta_2, \delta_1, \delta_2)$. Bosco's rule is denoted by $(5, 34, 45, 34, 58)$. Similarly, Life is one of the numerous range 1 LtL rules and is denoted by $(1, 3, 3, 3, 4)$.

Next we present definitions which are needed for the sliding block memory. These definitions for the most part conform to Life's definitions defined in the *Life Lexicon* [10]. The terminology in the remaining part of the paper also conforms to that used by the group of researchers devoted to the study of Life. For instance, this is where such terminology as "bug gun," "salvo of bugs," "shotgun," and so on originated.

Definition 1. *A* still life *is a configuration Λ which is a fixed point for $\mathcal{T}$. That is, $\mathcal{T}(\Lambda) = \Lambda$.*

Example 1 *Life's* block *is a 2×2 configuration of live sites that remains fixed as the rule updates. Similarly, Bosco's rule has a block, which is a fixed 6×6 configuration of live sites. We have generalized the block to LtL rules with arbitrarily large ranges [6].*

The block is the static piece used in the sliding block memory to store a register's value. It is used because it is a very commonly occurring still life; other shapes could also have been used.

Definition 2. *An* oscillator *or* periodic object *is a finite configuration Λ for which there exists a positive integer n so that $\mathcal{T}^n(\Lambda) = \Lambda$. The smallest such n is called the* period *of Λ.*

Example 2 *Bosco is the period 166 oscillator for which the rule is named (see A.mcl[1]). Bosco's trajectory is depicted in Figure 1: starting at time 0, moving northeast along the diagonal to the phase depicted at time 25 and then turning around at time 50 to move southwest along the diagonal. Observe that Bosco's phases at times 83 and 108 are rotated translations of those that are depicted at times 0 and 25, respectively. At time 133, a rotated translation of Bosco's time 50 turn would appear, but depicted instead is time 137, which is Bosco along with the spark (see Figure 2). At time 166 Bosco is back to the starting position.*

Definition 3. *A* spark *is a pattern that dies. The term is typically used to describe a collection of cells periodically thrown off by an oscillator or bug [10].*

[1] In the companion website [9], java applets which illustrate the constructions may be viewed in real-time. Here and after, references to these applets will be denoted by *.mcl. See the website for a more complete description of the applets.

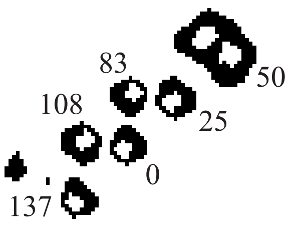


Fig. 1. Times 0, 25, 50, 83, 108, and 137 of Bosco's trajectory (A.mcl).

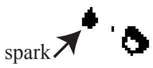


Fig. 2. The large arrow is pointing to Bosco's spark, which appears northwest of Bosco. The two adjacent cells closer to Bosco are also part of the spark.

A spark is useful for obvious reasons: it may interact with other live sites without affecting the oscillator that generates it. Bosco's spark is located quite a distance away, which is key to the numerous reactions needed for the sliding block memory.

Definition 4. *A* bug *is a finite configuration Λ for which there exists a finite time, τ, and a nonzero displacement vector, $\boldsymbol{d} = (d_1, d_2)$, such that $\mathcal{T}^\tau(\Lambda) = \Lambda + \boldsymbol{d}$. The smallest such τ is a bug's* period*, mod translation, in the direction of $\boldsymbol{d}$.*

Definition 5. *The* speed *of a bug is $max(|d_1|, |d_2|)/\tau$.*

LtL's bugs are generalizations of Life's famous spaceships. They are characterized according to their trajectories and their speeds. The two bug types used in the construction of the sliding block memory are speed 5/6 *orthogonal bugs*, which have displacement vectors $\boldsymbol{d} = (5, 0)$ and period $\tau = 6$, so they move 5 cells every 6 time steps, and speed 8/16 *diagonal bugs*, which have displacement vectors $\boldsymbol{d} = (8, 8)$ and period $\tau = 16$, so they move in the diagonal direction 8 cells every 16 time steps. Other bug varieties are defined in [6].

Example 3 *Bosco's rule supports bugs of varying shapes and speeds. In Figure 3 we illustrate the trajectory of a speed 5/6 orthogonal bug, which is the most crucial part of the sliding block memory. For additional examples, see B.mcl.*

Another crucial ingredient in the sliding block memory is the *tripling reaction*, in which Bosco turns a speed 5/6 orthogonal bug 90 degrees and creates another copy of Bosco as well as a speed 8/16 diagonal bug in the process (Figure 4 and C.mcl). The additional Bosco and diagonal bug can then be turned into speed 5/6 orthogonal bugs, hence the reaction's name.

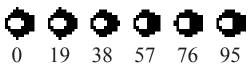


Fig. 3. Times 0, 19, 38, 57, 76, and 95 of the trajectory of a speed 5/6 orthogonal bug are depicted. These specific times were selected so that each of the bug's 6 phases may be viewed without overlap.

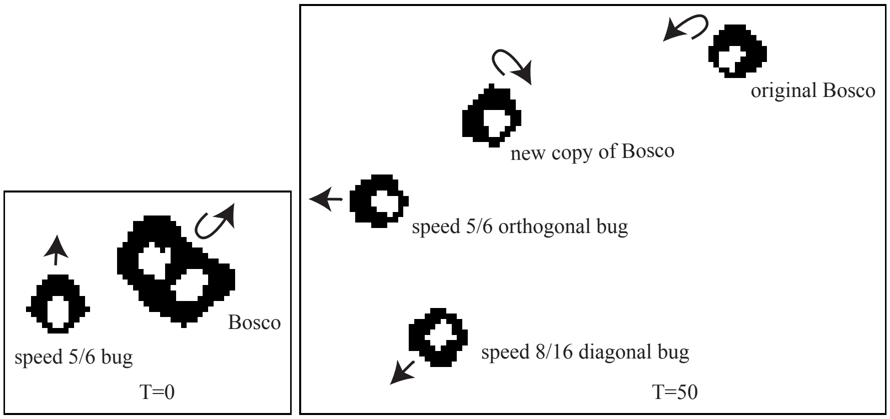


Fig. 4. Time 0 on the left shows Bosco and a speed 5/6 orthogonal bug heading north. After 50 time steps, which is depicted on the right, a figure, which will become a speed 5/6 bug turned 90 degrees from the original, appears along with figures that will become a new copy of Bosco and a period 16 diagonal bug, respectively (C.mcl).

In order to prove that Life is universal, Conway needed first to show that a finite population of live sites could generate an infinite population. This challenge was posed in Martin Gardner's Mathematical Games column in Scientific American in 1970 [1]. William Gosper won the fifty dollar prize attached to the challenge when he constructed Life's first *glider gun* (Gosper.mcl). Similarly, we needed bug guns for Bosco's rule and Hickerson led the way by constructing the first one which, like Bosco, has period 166 (D.mcl). Due to space-time considerations, the sliding block memory has period 332. We thus had to build special period 332 guns like the one depicted in Figure 5. The gun works as follows: Two copies of Bosco, denoted by B_i and B_{ii} create one speed 5/6 orthogonal bug every 166 time steps. One such bug is depicted in the figure and denoted by b_i. Meanwhile, two more copies of Bosco, denoted by B_{iii} and B_{iv} form a stable block every 166 time steps. When b_i collides with the block, b_i is turned 180 degrees and crashes into the next bug, b_{ii}, created by B_i and B_{ii}. Both bugs are annihilated in the process. The next block formed by B_{iii} and B_{iv} thus remains fixed until the Bosco denoted by B_v transforms it into a speed 5/6 orthogonal bug. One such bug, b_v is depicted. This occurs once every 332 time steps, resulting in a period 332 gun (E.mcl).

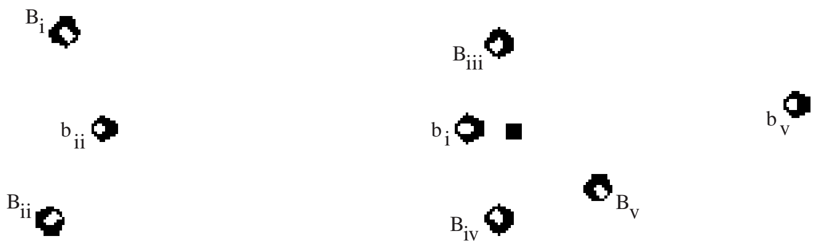


Fig. 5. Period 332 bug gun (E.mcl).

In addition to the standard period 332 gun, we also needed guns whose outputs are turned 90 degrees and/or shifted. Through extensive experimentation we found that strategically placed copies of Bosco can turn, shift, and adjust the phase of a speed 5/6 bug. Thus, by adding appropriately placed Boscos, we were able to construct the required guns (eg. F.mcl).

3 Bosco's Sliding Block Memory

In the auxiliary storage device designed by Conway and mentioned in the introduction, each *register* contains a block (see Example 1), whose distance from the computer (on a certain scale) indicates the number the register contains. *Salvos* of gliders are used to *push* a block (increase the contents of a register by 1) or *pull* a block (decrease the contents of a register by 1). Another glider is used to *test* whether a register's contents are 0. Conway found a salvo of 2 gliders that could *pull* a block a diagonal distance of 3 and a salvo of 30 gliders that could *push* a block a diagonal distance of 3. A diagonal distance of 3 is thus used to represent a change of 1 in a register. Conway argued that his design was sufficient to prove that Life is universal since Minsky has shown that a finite computer with two such memory registers is sufficient to simulate a universal Turing machine [3], [11]. In 1990 Hickerson designed and built Life's *Sliding Block Memory* (SBM), which simplifies the design described by Conway [7].

We used Conway's idea for a register and the simplifications made by Hickerson to build an SBM for Bosco's rule. We found a 5-bug salvo that pushes a block 10 units and a 6-bug salvo that pulls a block 10 units. Thus, a horizontal distance of 10 represents a change of 1 in a register. All of the bugs in the push and pull salvos are speed 5/6 so that the distance representing a change of 1 in a register must be a multiple of 5. A distance of 10 was chosen since it is the smallest multiple of 5 for which all of the moving parts of the SBM would interact synergistically without destroying one another.

Bosco's SBM consists of a shotgun which produces every 332 time steps an 11-bug salvo containing both the push and pull salvos. There is a control device which releases the push and pull salvos when instructed to do so by external circuitry. There is also a *test for zero*, which automatically reports when a block is pulled from 1 to 0. Let us describe the parts of the SBM in detail.

The 6-bug pull salvo is depicted in Figure 6 and works as follows: A block begins in position $(0,0)$. Bug b_1 shifts it to $(-20,1)$ by time 57; b_2 then shifts it to $(-16,-12)$ by time 86; b_3 shifts it to $(-13,-6)$ by time 112, while b_4 annihilates the debris created in the reaction, by time 126. Bug b_5 then shifts it to $(-10,0)$ by time 208 while b_6 cleans up the debris and at time 222 all that remains is the block, which has been pulled 10 units (G.mcl).

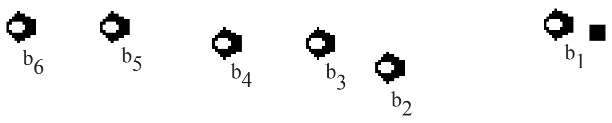


Fig. 6. A 6-bug salvo pulls a block 10 units (G.mcl).

The 5-bug push salvo is depicted in Figure 7 and works as follows: A block begins in position $(0,0)$. Bug b_7 transforms it into a still life by time 108, which b_8 transforms into two blocks by time 183, one of which is in position $(44,0)$. The other, which is in position $(35,48)$, is annihilated when b_{10} collides with it. Meanwhile b_9 shifts the other block to position $(27,5)$ by time 246. Bug b_{11} then shifts the block to $(10,0)$ by time 320. At time 327 debris created during the reaction has died and all that remains is a block, pushed 10 units from its original location (H.mcl).

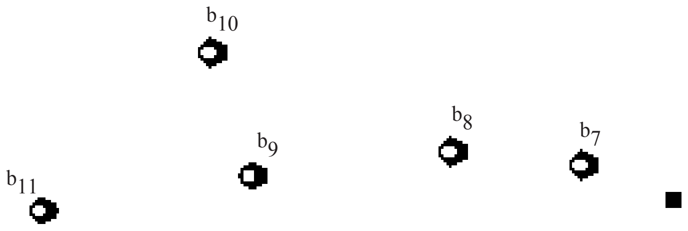


Fig. 7. A 5-bug salvo pushes a block 10 units (H.mcl).

Of course, numerous salvos of bugs can be positioned to push and pull the block various distances; however, in the other more efficient cases we tried, the test for zero, which we describe next, failed.

To test whether a register is 0, Conway designed a test which destroyed the block and then had to rebuild it. Hickerson designed a simpler test for Life's SBM, which does not destroy the block in the process; instead whenever the block is pulled from 1 to 0, this transition is reported to the computer. Our test for zero is like Hickerson's. It is depicted in Figure 8 and works as follows: Every 332 time steps, a bug is fired from the gun denoted by $G_{zerodetector}$. If the block is pulled from position from 1 to 0, this bug is annihilated in the process, without harming the block. Otherwise, the bug moves south unharmed (I.mcl).

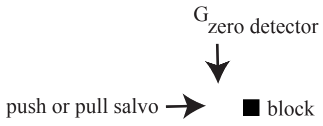


Fig. 8. The test for zero automatically reports when the block is pulled from 1 to 0.

A sketch of the shotgun is depicted in Figure 9 and the 11-bug salvo which it creates is in Figure 10. The shotgun consists of 3 p166 guns, denoted by g_3, g_4, and g_5 which create bugs b_3, b_4, and b_5. Meanwhile, gun G annihilates all 3 bugs every 332 time steps, resulting in the 3 bugs being created once every 332 time steps. Two more period 332 guns, denoted by G_{11} and G_2 (which include bug shifting reactions and a ninety degree turn so the new bugs do not collide with those already created) are located south and further east and create 2 more of the salvo's bugs, b_{11} and b_2, respectively. One more such gun, G_1 appears even further east and five more such guns G_6, G_9, G_8, G_7, and G_{10} appear to the northeast and they produce the remaining 6 bugs of the salvo, b_1, b_6, b_9, b_8, b_7, and b_{10}, respectively. All bugs fall into formation to head east (J.mcl).

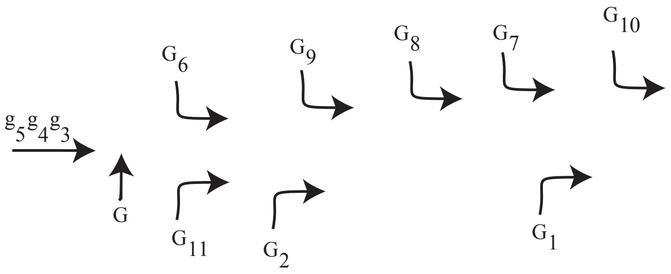


Fig. 9. Shotgun for Bosco's SBM: creates the 11-bug salvo depicted in Figure 10.

Since the push and pull salvos are created together by the shotgun, one, the other, or both must be suppressed so that the block is either pushed (if only the

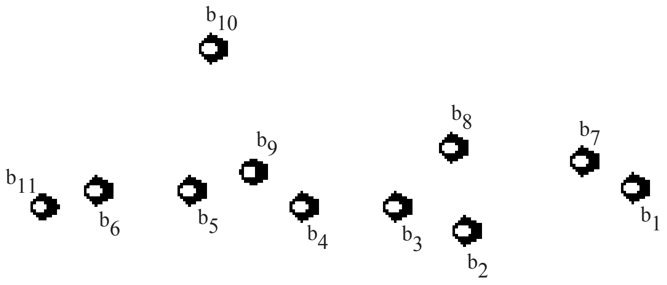


Fig. 10. Salvo created by the shotgun depicted in Figure 9. This 11-bug salvo is a disjoint union of the push and pull salvos depicted in Figures 6 and 7.

pull is suppressed), pulled (if only the push is suppressed), or neither (if both are suppressed). Push (or increment) and pull (or decrement) suppressors were designed for this job, both consist of 3 period 332 guns. The following describes how the push suppressor works.

The *push suppressor* is depicted in Figure 11 and works as follows: Three guns, G_A, G_B, and G_C send "suppressor" signal bugs A, B, and C toward the 11-bug salvo. Bug A collides with b_{11} causing both to vanish. Bug B collides with b_8 to form a block which remains fixed until b_9 crashes into it resulting in the annihilation of both. Bug C collides with b_7 to create a period 16 diagonal bug heading northwest, which collides with b_{10} and both are annihilated in the process. All 5 bugs, b_i, $i = 7, 8, 9, 10$, and 11 of the push salvo are thus annihilated by the output of the three guns, while the 6 bugs of the pull salvo remain unchanged (K.mcl).

The *pull suppressor* is depicted in Figure 12 and works as follows: Three guns, G_D, G_E, and G_F send "suppressor" signal bugs D, E, and F toward the 11-bug salvo. Bug D collides with b_5, causing a reaction that results in the annihilation of D, b_5, and b_6. Bug E collides with b_2 and both bugs are annihilated. Bug F turns b_1 into a block which remains fixed until b_3 crashes into it, causing a reaction that annihilates b_3 and b_4. All 6 bugs, b_i, $i = 1 - 6$ of the pull salvo are thus annihilated by the output of the three guns, while the 5 bugs of the push salvo remain unchanged (L.mcl).

An external signal is used to indicate whether the block is to be pushed or pulled. If no such signal is sent, the block remains fixed. A push is signaled by annihilating the three bugs from the push suppressor. A control device is needed to do this. The push suppressor and control device are depicted in Figure 11 and work as follows: Together guns G_A, G_B, and G_C suppress the push salvo. Meanwhile, a control device, consisting of three copies of Bosco, B_1, B_2, and B_3 wait for an external input, which is a speed 5/6 orthogonal bug, denoted by b_{push} that will signal a push as follows: b_{push} goes into Bosco's tripling reaction, denoted by B_1, the output of which are bugs b_B and b_C (after an additional copy of Bosco, denoted by B_2 transforms the tripling reaction's period 16 diagonal bug into b_C and another copy of Bosco, B_3 annihilates the tripling reaction's unwanted copy of Bosco). Bug b_B annihilates the bugs from guns A and B and bug B_C annihilates the bug from gun C. The result is that a push salvo survives (M.mcl).

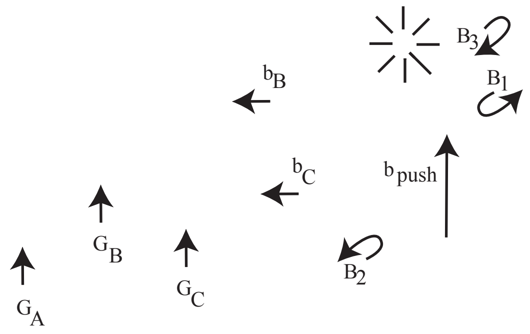


Fig. 11. Push suppressor, G_A, G_B, and G_C, along with control device B_1, B_2, and B_3, that waits for an external push signal, b_{push} (M.mcl).

Similarly, a pull is signaled by annihilating the three bugs from the pull suppressor. The pull suppressor and required control device are depicted in Figure 12 and work as follows: Together guns G_D, G_E, and G_F suppress the pull salvo. Meanwhile, the control device, consisting of nine copies of Bosco, $B_4 - B_{12}$ wait for an external input, which is a speed 5/6 orthogonal bug, denoted by b_{pull}, that signals a pull as follows: b_{pull} goes into Bosco's tripling reaction, denoted by B_4, the output of which are bugs b_D, b_E, and b_F (after the three copies of Bosco denoted by B_5, B_6, and B_7 transform B_4's period 16 diagonal bug output into b_D, Boscos B_8, B_9, and B_{10} transform B_4's Bosco output into b_F, and Boscos B_{11} and B_{12} shift B_4's speed 5/6 orthogonal bug output so that it is in B_E's position. Note that the placement of the Boscos is crucial since all 9 of them must do their individual jobs without interacting with one another or any of the other moving parts of the pull signal). Bug b_D annihilates the bug from gun D, bug b_E annihilates the bug from gun E, and bug b_F annihilates the bug from gun F. The result is that a pull salvo survives (N.mcl).

In order to test Bosco's SBM, we created an experiment that includes an external circuit, consisting of four copies of Bosco, denoted by $B_{13} - B_{16}$, one period 332 gun, denoted by $G_{pushsignal}$, and one period 4980 gun, denoted by G_{p4980}. This system test also includes the shotgun, push and pull suppressors and their control devices, as well as the test for zero. It is depicted in Figure 13 and works as follows: The gun, $G_{zerodetector}$, outputs one bug every 332 time steps. If the block is *not* pulled from 1 to 0, Boscos B_{13}, B_{14}, and B_{15} turn this bug 90 degrees each, as denoted by the arrows, and so that it annihilates the bug from the period 332 gun $G_{pushsignal}$ which, if not stopped would signal a push as described above. If the block is pulled from 1 to 0, the output bug from $G_{zerodetector}$ is instead annihilated and thus a push signal arrives 4648 time steps

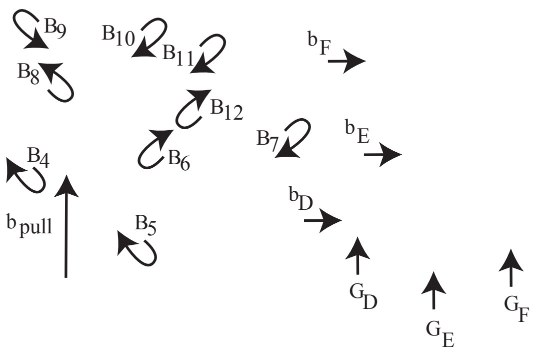


Fig. 12. Pull suppressor, G_D, G_E, and G_F, along with control device $B_4 - B_{12}$ that waits for an external pull signal, b_{pull} (N.mcl).

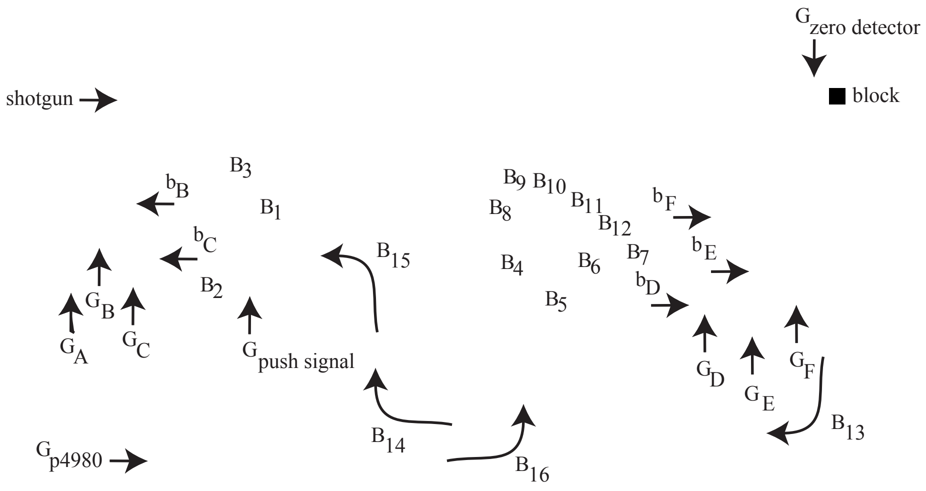


Fig. 13. SBM system test (O.mcl).

later (the time length is a consequence of the particular spatial configuration). Meanwhile, the gun G_{p4980} outputs one bug every 4980 time steps and the Bosco denoted by B_{16} turns this bug 90 degrees so that it becomes a pull signal every 4980 generations. In the experiment (O.mcl), a block begins in position 1. It is pushed (10 cells to the right) and then pulled (10 cells to the left). After that, nothing happens until the first bug from G_{p4980} arrives and signals a pull. This pulls the block from 1 to 0 so that one bug from gun $G_{zerodetector}$ is annihilated and 4648 time steps later a push is signaled. This pattern repeats, so that the block is pulled to 0 and then pushed to 1, ad infinitum.

4 Conclusion

We have shown that Bosco's rule supports various constructions, including the sliding block memory, which is a register that can store any nonnegative integer. The SBM has three basic operations: increment, decrement, and test for zero. Minsky has shown that just 2 such registers suffice to simulate a universal Turing machine [11].

In a companion paper we describe LtL rules similar to Bosco's (from various ranges) which are candidates for universality. We explore questions which have arisen since Conway challenged us to do these constructions, including whether the details given in his proof that Life is universal are sufficient and what necessary and sufficient conditions are required to prove that Bosco's rule, or any two-dimensional cellular automaton, is universal. For instance, is it necessary to explicitly construct all reactions and a sliding block memory (or something equivalent)? Is it necessary to build a *Universal Minsky Register Machine* (or something equivalent), like the one Paul Chapman built for Life? [12] Or is there a set of axioms one can state that ensures that a rule which satisfies the axioms is universal? If so, what is the smallest set of axioms?

5 Acknowledgements

Thanks to: Dean Hickerson for his interest in Bosco's rule, insights, contributions, and feedback; David Griffeath and Cris Moore for organizing the *New Constructions in Cellular Automata* conference at the Santa Fe Institute in 1998; Joan Wheeler for editorial assistance; and Carl Addicott for perspective.

References

1. Gardner, M.: Mathematical games–the fantastic combinations of John Conway's new solitaire game, Life, Sci. Am. **223** (1970) 120–123
2. Summers, J.: Game of Life status page, entropymine.com/jason/life/status.html
3. Berlekamp, E., Conway, J., Guy, R.: What is Life? in: *Winning Ways for Your Mathematical Plays*, vol. 2, Academic Press, New York, 1982, Chapter 25
4. Griffeath, D.: Self-organization of random cellular automata: four snapshots, in: *Probability and Phase Transitions*, D. Griffeath, 1994, (G. Grimmett Ed.), Kluwer Academic, Dordrecht/Norwell, MA
5. Evans, K.: *Larger than Life: it's so nonlinear*, Ph.D. dissertation, University of Wisconsin - Madison, 1996, http://www.csun.edu/∼kme52026/thesis.html
6. Evans, K.: Threshold-range scaling of Life's coherent structures, Physica D **183** (2003) 45-67
7. Hickerson, D.: Description of sliding block memory, 1990, http://www.radicaleye.com/lifepage/patterns/sbm/sbm.html
8. Griffeath, D.: Primordial Soup Kitchen, http://psoup.math.wisc.edu/kitchen.html
9. Evans, K.: http://www.csun.edu/∼kme52026/bosco/bosco.html
10. Silver, S.: Life Lexicon Home Page, www.argentum.freeserve.co.uk/lex_home.htm
11. Minsky, M.: Computation: Finite and Infinite Machines, Pren. Hall, 1967, Sec. 15.1
12. Chapman, P.: Life Universal Computer, 2002, http://www.igblan.free-online.co.uk/igblan/ca/

Sequential P Systems with Unit Rules and Energy Assigned to Membranes

Rudolf Freund[1], Alberto Leporati[2], Marion Oswald[1], and Claudio Zandron[2]

[1] Faculty of Informatics, Vienna University of Technology
{rudi,marion}@emcc.at

[2] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano - Bicocca
{leporati,zandron}@disco.unimib.it

Abstract. We introduce a new variant of membrane systems where the rules are directly assigned to membranes (and not to the regions as this is usually observed in the area of membrane systems) and, moreover, every membrane carries an energy value that can be changed during a computation by objects passing through the membrane. For the application of rules leading from one configuration of the system to the succeeding configuration we consider a sequential derivation mode and do not use the mode of maximal parallelism. The result of a successful computation is considered to be the distribution of energy values carried by the membranes. We show that for such systems using a kind of priority relation on the rules we already obtain universal computational power. When omitting the priority relation, we obtain a characterization of the family of Parikh sets generated by context-free matrix grammars (with λ-rules).

Keywords: computational completeness, matrix grammars, membrane computing, P systems

1 Introduction

In 1998 Gheorghe Păun introduced membrane systems (in [12]) as distributed and parallel computing devices that were abstracted from the biological functioning of living cells. For motivations and examples as well as for further details we refer to [13]; for actual developments in the area of P systems see [17].

Considering the energy balancing of processes in a cell first was investigated in [14] and then in [4]. There the energies of all rules to be used in a given step in a membrane are summed up; if the total amount of energies is positive ([14]) or within a given range ([4]), then this multiset of rules can be applied if it is maximal with this property.

We here take another approach. In contrast to most models of P systems where the evolution rules are placed within a region, in this paper we consider membrane systems where the rules are directly assigned to the membranes (as already done in [7]) and have to be applied in a sequential way (for sequential variants of P systems see, for example, [2] and [3]). Moreover, each membrane carries an energy value. As long as the energy value of a membrane is non-negative, by a rule application, singleton objects can be rewritten while passing through membranes, thereby consuming or producing energy that is added to or subtracted from the energy value of the respective membrane. We also consider a kind of priority relation on the rules assigned to the membranes by choosing the one first that changes the energy value of the membrane under consideration in a maximal way. The result of a successful computation is stored in the final energy values of the membranes.

In the following section we first give some preliminary definitions and recall some notions and results for register machines and matrix grammars, the computation models we use for proving the results elaborated in this paper; then we introduce P systems with unit rules and energy assigned to membranes. In the third section we show that when using a kind of priority among the rules, the introduced systems can simulate register machines quite easily, which proves their computational completeness. A characterization of $PsMAT^{\lambda}$ is obtained when omitting the priority relation.

2 Definitions

The set of non-negative integers is denoted by $\mathbf{N}$. An *alphabet* V is a finite non-empty set of abstract *symbols*. Given V, the free monoid generated by V under the operation of concatenation is denoted by V^*; the *empty string* is denoted by λ, and $V^* - \{\lambda\}$ is denoted by V^+. By $\mid x \mid$ we denote the length of the word x over V.

Let $\{a_1, ..., a_n\}$ be an arbitrary alphabet; the number of occurrences of a symbol a_i in x is denoted by $\mid x \mid_{a_i}$; the *Parikh vector* associated with x with respect to $a_1, ..., a_n$ is $(\mid x \mid_{a_1}, ..., \mid x \mid_{a_n})$. The *Parikh image* of a language L over $\{a_1, ..., a_n\}$ is the set of all Parikh vectors of strings in L. For a family of languages FL, the family of Parikh images of languages in FL is denoted by $PsFL$. A (finite) multiset $\langle m_1, a_1 \rangle ... \langle m_n, a_n \rangle$ with $m_i \in \mathbf{N}$, $1 \leq i \leq n$, is represented as any string x the Parikh vector of which with respect to $a_1, ..., a_n$ is $(m_1, ..., m_n)$.

The family of recursively enumerable languages is denoted by RE, the family of context-free languages by CF. For more details on formal language theory we refer to [16] as well as to [13].

2.1 Register Machines

A *deterministic register machine* is a construct $M = (m, R, l_0, l_h)$, where m is the number of registers, R is a finite set of instructions injectively labelled with elements from a given set $lab(M)$, l_0 is the initial/start label, and l_h is the final label.

The instructions are of the following forms:

- $l_1 : (ADD(r), l_2)$
  Add 1 to the contents of register r and proceed to the instruction (labelled with) l_2. (We say that we have an add-instruction.)
- $l_1 : (SUB(r), l_2, l_3)$
  If register r is not empty, then subtract 1 from its contents and go to instruction l_2, otherwise proceed to instruction l_3. (We say that we have a subtract-instruction.)
- $l_h : Halt$
  Stop the machine. The final label l_h is only assigned to this instruction.

The results proved in [6] (based on the results established in [11]) as well as in [8] and [9] immediately lead to the following result:

Proposition 1. *For any partial recursive function $f : \mathbf{N}^\alpha \to \mathbf{N}^\beta$ there exists a deterministic $(\max\{\alpha, \beta\} + 2)$-register machine M computing f in such a way that, when starting with $(n_1, ..., n_\alpha) \in \mathbf{N}^\alpha$ in registers 1 to α, M has computed $f(n_1, ..., n_\alpha) = (r_1, ..., r_\beta)$ if it halts in the final label h with registers 1 to β containing r_1 to r_β, and all other registers being empty; if the final label cannot be reached, $f(n_1, ..., n_\alpha)$ remains undefined.*

2.2 Matrix Grammars

A context-free *matrix grammar* (without appearance checking) is a construct $G = (N, T, S, M)$ where N and T are sets of *non-terminal* and *terminal symbols,* respectively, with $N \cap T = \emptyset$, $S \in N$ is the *start symbol*, M is a finite set of *matrices*, $M = \{m_i \mid 1 \leq i \leq n\}$, where the matrices m_i are sequences of the form $m_i = (m_{i,1}, \ldots, m_{i,n_i})$, $n_i \geq 1$, $1 \leq i \leq n$, and the $m_{i,j}$, $1 \leq j \leq n_i$, $1 \leq i \leq n$, are context-free productions over (N, T). For $m_i = (m_{i,1}, \ldots, m_{i,n_i})$ and $v, w \in (N \cup T)^*$ we define $v \Longrightarrow_{m_i} w$ if and only if there are $w_0, w_1, \ldots, w_{n_i} \in (N \cup T)^*$ such that $w_0 = v$, $w_{n_i} = w$, and for each $j, 1 \leq j \leq n_i$, w_j is the result of the application of $m_{i,j}$ to w_{j-1}. The language generated by G is

$$\begin{aligned} L(G) = \{w \in T^* \mid\ & S \Longrightarrow_{m_{i_1}} w_1 \ldots \Longrightarrow_{m_{i_k}} w_k,\ w_k = w, \\ & w_j \in (N \cup T)^*,\ m_{i_j} \in M \ \text{ for } 1 \leq j \leq k, k \geq 1\}. \end{aligned}$$

According to the definitions given in [1], the last matrix can already finish with a terminal word without having applied the whole sequence of productions. The family of languages generated by matrix grammars without appearance checking is denoted by MAT^λ. It is known that $PsCF \subset PsMAT^\lambda \subset PsRE$. Further details about matrix grammars can be found in [1] and in [16].

2.3 P Systems with Unit Rules and Energy Assigned to Membranes

A *P system with unit rules and energy assigned to membranes of degree* $d+1$ is a construct Π of the form

$$\Pi = (O, \mu, e_0, ..., e_d, w_0, ..., w_d, R_0, ..., R_d)$$

where

- O is an alphabet of *objects*;
- μ is a *membrane structure* (with the membranes labelled by numbers $0, ..., d$ in a one-to-one manner);
- $e_0, ..., e_d$ are the initial energy values assigned to the membranes $0, ..., d$;
- $w_0, ..., w_d$ are multisets over V associated with the regions $0, ..., d$ of μ;
- $R_0, ..., R_d$ are finite sets of *unit rules* associated with the membranes $0, ..., d$, which are of the form $(\alpha : a, \Delta e, b)$ where $\alpha \in \{in, out\}$, $a, b \in O$, and $|\Delta e|$ is the amount of energy that - for $\Delta e \geq 0$ - is added to or - for $\Delta e < 0$ - is subtracted from e_i (the energy assigned to membrane i) by the application of the rule.

Instead of $(\alpha : a, \Delta e, b) \in R_i$ we will also write $(\alpha_i : a, \Delta e, b)$ and then, instead of $R_0, ..., R_d$, specify only one set of rules R with

$$R := \{(\alpha_i : a, \Delta e, b) \mid (\alpha : a, \Delta e, b) \in R_i, 0 \leq i \leq d\}.$$

Starting from the *initial configuration*, which consists of μ, $e_0, ..., e_d$, and $w_0, ..., w_d$, the system passes from one configuration to another one by non-deterministically choosing one rule from R and applying it in the following sense (observe that here we consider a sequential mode for the application of only one rule instead of applying rules in a maximally parallel way as it is often required in P systems): applying $(in_i : a, \Delta e, b)$ means that an object a (being in the membrane immediately outside of i) is changed into b while entering membrane i thereby changing the energy value e_i of membrane i by Δe. On the other hand, the use of a rule $(out_i : a, \Delta e, b)$ changes object a into b while it passes out from membrane i changing its energy value by Δe. Yet the rules are only applicable if the amount e_i of energy assigned to membrane i fulfills the requirement $e_i + \Delta e \geq 0$; moreover, we use some sort of local priorities: if there is more than one rule associated with membrane i which could be applied, then one of the rules with $\max |\Delta e|$ has to be used.

A sequence of transitions is called a *computation*; it is *successful* if and only if it halts. The *result* of a successful computation is considered to be the distribution of energies among the membranes (a non-halting computation does not produce a result). Observe that in this model we do not take into account the environment.

3 Results

The following theorem establishes computational completeness for the new variant of sequential P systems introduced in this paper:

Theorem 1. *Each partial recursive function $f : \mathbf{N}^{\alpha} \to \mathbf{N}^{\beta}$ can be computed by a P system with unit rules and energy assigned to membranes with (at most)* $\max\{\alpha, \beta\} + 3$ *membranes.*

Proof. Consider a (deterministic) register machine $M = (m, P, 1, n)$ with m registers such that with the program P the function f is computed; the initial instruction has the label 1 and the halting instruction has the label n. Observe that according to the result stated in Proposition 1, $m = \max\{\alpha, \beta\} + 2$ is enough.

The input values $x_1, ..., x_\alpha$ are expected to be in the first α registers and the output values from $f(x_1, ..., x_\alpha)$ are expected to be in registers 1 to β at the end of a successful computation. Moreover, without loss of generality, we may assume that at the beginning of a computation all the registers except eventually the registers 1 to α contain zero.

We construct the P system

$$\begin{aligned}
\Pi &= (O, \mu, e_0, ..., e_m, w_0, ..., w_m, R),\\
O &= \{p_j, \widetilde{p_j} | 1 \leq j \leq n, j \in Lab(M)\},\\
\mu &= [_0[_1]_1 \cdots [_\alpha]_\alpha \cdots [_m]_m]_0,\\
e_i &= x_i \qquad \text{for } 1 \leq i \leq \alpha,\\
&\quad 0 \qquad \text{for } \alpha + 1 \leq i \leq m,\\
w_0 &= p_1,\\
w_i &= \lambda \qquad \text{for } 1 \leq i \leq m,\\
R &= \{(in_i : p_j, 1, \widetilde{p_j}), (out_i : \widetilde{p_j}, 0, p_k) \mid j : (ADD(i), k) \in P\}\\
&\cup \{(in_i : p_j, 0, \widetilde{p_j}), (out_i : \widetilde{p_j}, -1, p_k), (out_i : \widetilde{p_j}, 0, p_l) \mid\\
&\quad j : (SUB(i), k, l) \in P\}.
\end{aligned}$$

The contents of register i, $1 \leq i \leq m$, is represented by the energy value e_i of membrane i.

The set of rules R depends on the instructions of P; in more detail, the simulation works as follows:

1. Each add-instruction $j : (ADD(i), k, k) \in P$, $1 \leq i \leq m$, is simulated in two steps by using the rules $(in_i : p_j, 1, \widetilde{p_j})$ and $(out_i : \widetilde{p_j}, 0, p_k)$.
2. Each conditional subtract-instruction $j : (SUB(i), k, l) \in P$ is simulated in two steps by the rules $(in_i : p_j, 0, \widetilde{p_j})$ as well as $(out_i : \widetilde{p_j}, -1, p_k)$ or $(out_i : \widetilde{p_j}, 0, p_l)$.
   The condition of priority guarantees that $(out_i : \widetilde{p_j}, -1, p_k)$ is applied as long as e_i has a positive value. Only if in the current configuration $e_i = 0$, i.e., register i is empty, the rule $(out_i : \widetilde{p_j}, 0, p_l)$ can be used.

It follows from the description given above that after each simulation of an instruction each energy value e_i equals the contents of register i, $1 \leq i \leq m$. Hence, after having simulated the instruction *Halt* and halting the system by just doing nothing with the halting symbol p_n any more, the energy values $e_1, .., e_m$ equal the output of the program P. The only object remaining within the system is the final label p_n in region 0. □

On the other hand, when omitting the priority feature, we do not get systems with universal computational power. Let $PsPE_*(unit)$ denote the family of sets of Parikh vectors generated by P systems with unit rules and energy assigned to membranes without priorities and with an arbitrary number of membranes. The following two lemmas prove that $PsPE_*(unit) = PsMAT^\lambda$, i.e., we get a characterization of $PsMAT^\lambda$ by the new family $PsPE_*(unit)$.

Lemma 1. $PsPE_*(unit) \supseteq PsMAT^\lambda$

Proof. Let $G = (N, T, S, M)$ be a matrix grammar with λ-rules with every matrix being of the form $m_i = (m_{i,1}, \ldots, m_{i,n_i})$, $1 \leq i \leq n$, where $m_{i,j} = A_{i,j} \rightarrow w_{i,j,1} \ldots w_{i,j,n_{i,j}}$. Without loss of generality, we may assume that $n_{i,j} \leq 2$. Then we construct a P system Π with unit rules and energy assigned to membranes that simulates G as follows:

We label the skin membrane by 0 and for all elements B_i in $N \cup T$ we take a membrane labelled by i, $1 \leq i \leq m$, where $m = card(N \cup T)$ and $m' = card(T)$; moreover, we define a bijective function $index : \{1, ..., m\} \rightarrow N \cup T$ such that the terminal symbols have the indices 1 to m' and the start symbol S has the label m. Initially, every membrane has the energy value 0, i.e., $e_j = 0$ for $0 \leq j \leq m$.

Before starting the simulation of the matrices, we first have to add an additional step in order to get $e_m = 1$ as well as to have a non-deterministic choice for m_i by taking the rules $(in_m : p_0, 1, \widetilde{p_0})$ as well as $(out_m : \widetilde{p_0}, 0, p_{i,1,0})$ for every i with $1 \leq i \leq n$.

For the simulation of $m_{i,j}$, $1 \leq j \leq n_{i,j}$, $1 \leq i \leq n$, we have to take the following rules:

1. $\left(in_{index(A_{i,j})} : p_{i,j,0}, 0, \widetilde{p_{i,j,0}}\right)$ and $\left(out_{index(A_{i,j})} : \widetilde{p_{i,j,0}}, -1, \alpha_{i,j}\right)$ with
   - $\alpha_{i,j} \in \{p_{k,1,0} | 1 \leq k \leq n\}$ for $w_{i,j} = \lambda$ and $j = n_i$,
   - $\alpha_{i,j} = p_{i,j+1,0}$ for $w_{i,j} = \lambda$ and $j < n_i$,
   - $\alpha_{i,j} = p_{i,j,1}$ otherwise.
2. $\left(in_{index(w_{i,j,1})} : p_{i,j,1}, 1, \widetilde{p_{i,j,1}}\right)$ and $\left(out_{index(w_{i,j,1})} : \widetilde{p_{i,j,1}}, 0, \beta_{i,j}\right)$ with
   - $\beta_{i,j} \in \{p_{k,1,0} | 1 \leq k \leq n\}$ for $|w_{i,j}| = 1$ and $j = n_i$,
   - $\beta_{i,j} = p_{i,j+1,0}$ for $|w_{i,j}| = 1$ and $j < n_i$,
   - $\beta_{i,j} = p_{i,j,2}$ for $|w_{i,j}| = 2$.
3. $\left(in_{index(w_{i,j,2})} : p_{i,j,2}, 1, \widetilde{p_{i,j,2}}\right)$ and $\left(out_{index(w_{i,j,2})} : \widetilde{p_{i,j,2}}, 0, \gamma_{i,j}\right)$ with
   - $\gamma_{i,j} \in \{p_{k,1,0} | 1 \leq k \leq n\}$ for $j = n_i$,
   - $\gamma_{i,j} = p_{i,j+1,0}$ for $j < n_i$.

At some moment during the simulation of a derivation in the matrix grammar G by Π, we non-deterministically have to guess whether the current sentential form is already terminal (in order to be able to halt the computation in Π); for this purpose, we take the following rules:

1. $\left(out_{index(A_{i,j})} : \widetilde{p_{i,j,0}}, 0, p_f\right)$ can always be applied directly after having applied $\left(in_{index(A_{i,j})} : p_{i,j,0}, 0, \widetilde{p_{i,j,0}}\right)$; it allows us to finish the computation with the final object p_f if the current sentential form is terminal (i.e., $e_j = 0$ for $m' + 1 \leq j \leq m$).

2. $(in_j : p_f, -1, \widetilde{p_f})$ and $(out_j : \widetilde{p_f}, 0, \#)$ for $m' + 1 \leq j \leq m$ are used if the current sentential form has not been terminal (which means $e_j \neq 0$ for some j with $m' + 1 \leq j \leq m$) when introducing p_f; in that case we ensure that the system Π does not halt by entering an infinite loop with the trap symbol # using the following rules:
   $(in_m : \#, 0, \#)$ and $(out_m : \#, 0, \#)$.

If p_f cannot enter any of the membranes $m' + 1 \leq j \leq m$ this means that no non-terminal symbol occurs any more in the current sentential form of the simulated derivation in G, hence, it is correct to halt and thus to get the result stored in the values of e_j, $1 \leq j \leq m$, which by construction represents the corresponding result obtained by the simulated derivation in G. $\square$

Lemma 2. $PsPE_* (unit) \subseteq PsMAT^\lambda$

Proof. We first construct a matrix grammar which generates a suitable representation of all configurations reachable from the initial configuration in Π. Eliminating all non-final configurations from this set of reachable configurations by intersection with a regular language we obtain the set of halting configurations which immediately allows us to extract the terminal results by using a projection. As the family of matrix languages is closed under intersection with regular languages and projections (see [1]) this will prove the desired inclusion $PsPE_* (unit) \subseteq PsMAT^\lambda$.

We first start the construction of a matrix grammar $G = (V, T, M, S)$ generating the reachable configurations in Π for

$$\Pi = (O, \mu, e_0, ..., e_d, w_0, ..., w_d, R)$$

being an arbitrary P system with unit rules and energy assigned to membranes (arbitrary membrane structure, arbitrary number of membranes, arbitrary number of symbols). Taking $D = \{0, 1, ..., d\}$, we first define the mapping σ from the set of all possible configurations of Π to

$$(O \times D)^* \{D_0\} \{E_0\}^* ... \{D_d\} \{E_d\}^*$$

which for every configuration c of Π yields all its valid representations in such a way that:

- for every a in region i the symbol $(a, i) \in (O \times D)$ occurs in the string representation of c;
- the number of symbols (a, i) occurring in the string representation of c exactly coincides with the number of symbols a occurring in region i;
- $D_0 E_0^{e_0} ... D_d E_d^{e_d}$ is the second part of the string representation of configuration c with e_i, $0 \leq i \leq d$, being the energy value assigned to membrane i in c.

In G, we start with an initial matrix $[S \to s]$ such that s is a valid string representation of the initial configuration in the form defined above, i.e., $s \in \sigma(\textit{initial configuration})$.

The rules in R are simulated in the following way (by $[i]$ we denote the label of the membrane encapsulating membrane i):

- For a rule $(in_i : a, \Delta e, b)$ with $\Delta e \geq 0$ we take the matrix
  $\left[(a,[i]) \to (b,i)\,, D_i \to D_i E_i^{\Delta e}\right].$
- For a rule $(in_i : a, \Delta e, b)$ with $\Delta e < 0$ we take the matrix
  $\left[(a,[i]) \to \widetilde{(b,i)}, (E_i \to \lambda,)^{-\Delta e}\, \widetilde{(b,i)} \to (b,i)\right].$
  The notation $(E_i \to \lambda,)^n$, $n > 0$, is taken for a sequence of n productions $E_i \to \lambda$. We should like to recall the fact that the sequence of $-\Delta e$ rules $E_i \to \lambda$ is only applicable if the amount e_i of energy assigned to membrane i fulfills $e_i + \Delta e \geq 0$; hence we may be forced to stop in the middle of a matrix, because not enough energy is assigned to membrane i.
- For a rule $(out_i : a, \Delta e, b)$ with $\Delta e \geq 0$ we take the matrix
  $\left[(a,i) \to (b,[i])\,, D_i \to D_i E_i^{\Delta e}\right].$
- For a rule $(out_i : a, \Delta e, b)$ with $\Delta e < 0$ we take the matrix
  $\left[(a,i) \to \widetilde{(b,[i])}, (E_i \to \lambda,)^{-\Delta e}\, \widetilde{(b,[i])} \to (b,[i])\right].$

After the application of a matrix described above, we obtain a valid string representation of the configuration obtained from the previous configuration by applying the corresponding rule in Π. On the other hand, every string obtained from the (complete) application of a matrix to a valid string representation of a reachable configuration c is a valid string representation of the configuration resulting from the application of the corresponding rule in Π to c.

All the symbols introduced so far are non-terminal symbols. Except for the objects of the form $\widetilde{(b,j)}$ we now introduce the corresponding terminal symbol a_t for the non-terminal symbol a and we add the matrices $[a \to a_t]$. For later use in this proof, we also define the bijection t mapping each terminal symbol a_t (back) to the original symbol a; t is a renaming morphism, and obviously the family of matrix languages is closed under renamings.

Hence, in total we have obtained the matrix grammar

$$\begin{aligned}
G &= (N,T,S,M)\,,\\
N &= \{S\} \cup \{D_i, E_i \mid 0 \leq i \leq d\} \cup \left\{(a,i)\,, \widetilde{(a,i)} \mid a \in O, 0 \leq i \leq d\right\},\\
T &= \left\{a_t \mid a \in \left(N - \left\{\widetilde{(b,j)} \mid b \in O, 0 \leq j \leq d\right\}\right)\right\},\\
M &= \{[S \to s] \mid s \in \sigma(\textit{initial configuration})\}\\
&\quad \cup \left\{\left[(a,[i]) \to (b,i)\,, D_i \to D_i E_i^{\Delta e}\right] \mid (in_i : a, \Delta e, b) \in R, \Delta e \geq 0\right\}\\
&\quad \cup \left\{\left[(a,[i]) \to \widetilde{(b,i)}, (E_i \to \lambda,)^{-\Delta e}\, \widetilde{(b,i)} \to (b,i)\right] \mid \right.\\
&\qquad\quad \left.(in_i : a, \Delta e, b) \in R, \Delta e < 0\right\}\\
&\quad \cup \left\{\left[(a,i) \to (b,[i])\,, D_i \to D_i E_i^{\Delta e}\right] \mid (out_i : a, \Delta e, b) \in R, \Delta e \geq 0\right\}\\
&\quad \cup \left\{\left[(a,i) \to \widetilde{(b,[i])}, (E_i \to \lambda,)^{-\Delta e}\, \widetilde{(b,[i])} \to (b,[i])\right] \mid \right.\\
&\qquad\quad \left.(out_i : a, \Delta e, b) \in R, \Delta e < 0\right\}.
\end{aligned}$$

Due to the given construction, for $L(G)$ the following holds:

1. Every element in $L(G)$ represents a reachable configuration of Π.
2. If c is a reachable configuration in Π, then $L(G)$ contains a valid string representation of c.

Now we construct a regular set R describing the non-halting configurations of Π :

Let n be the total number of symbols (in the multiset sense) occurring in the initial configuration. Then $R = R_1 \cup R_2 \cup R_3 \cup R_4$ where

– R_1 is the (finite) union of all (regular) sets of the form
$$(O \times D)^{n_1} \{(a,i)\} (O \times D)^{n_2} \{D_0\} \{E_0\}^* \ldots \\ \{D_j\} \{E_j\}^* \ldots \{D_d\} \{E_d\}^*$$
such that $n_1 + n_2 + 1 = n$, $(in_j : a, \Delta e, b) \in R$, region i contains membrane j, and $\Delta e \geq 0$;

– R_2 is the (finite) union of all (regular) sets of the form
$$(O \times D)^{n_1} \{(a,i)\} (O \times D)^{n_2} \{D_0\} \{E_0\}^* \ldots \\ \{D_j\} \{E_j\}^{-\Delta e} \{E_j\}^* \ldots \{D_d\} \{E_d\}^*$$
such that $n_1 + n_2 + 1 = n$, $(in_j : a, \Delta e, b) \in R$, region i contains membrane j, and $\Delta e < 0$;

– R_3 is the (finite) union of all (regular) sets of the form
$$(O \times D)^{n_1} \{(a,j)\} (O \times D)^{n_2} \{D_0\} \{E_0\}^* \ldots \\ \{D_j\} \{E_j\}^* \ldots \{D_d\} \{E_d\}^*$$
such that $n_1 + n_2 + 1 = n$, $(out_j : a, \Delta e, b) \in R$, and $\Delta e \geq 0$;

– R_4 is the (finite) union of all (regular) sets of the form
$$(O \times D)^{n_1} \{(a,j)\} (O \times D)^{n_2} \{D_0\} \{E_0\}^* \ldots \\ \{D_j\} \{E_j\}^{-\Delta e} \{E_j\}^* \ldots \{D_d\} \{E_d\}^*$$
such that $n_1 + n_2 + 1 = n$, $(out_j : a, \Delta e, b) \in R$, and $\Delta e < 0$.

The set R is a finite union of regular sets, i.e., R is a regular set, too, and it describes the situations where a rule of Π is still applicable, hence, the non-halting configurations. Therefore, $(N^* - R)$ contains a lot of garbage, but also every string being a valid representation of a halting configuration.

Finally, let $p : N^* \to \{e_i \mid 1 \leq i \leq d\}^*$ be the projection mapping E_i to e_i (i.e., $p(E_i) = e_i$), $1 \leq i \leq d$, and erasing all other symbols ($p(X) = \lambda$ for all $X \in N - \{E_i | 1 \leq i \leq d\}$). In sum, we obtain

$$L(\Pi) = p(t(L(G)) \cap (N^* - R)),$$

i.e., (in the representation as multisets over $\{e_i \mid 1 \leq i \leq d\}$) $L(\Pi)$, the set of Parikh vectors generated by Π, is the projection of the intersection of the renaming of a matrix language with a regular set, hence, due to the closure properties of the family of matrix languages, $L(\Pi)$ is a matrix language, too, which observation concludes the proof. □

If we now combine the two previous lemmas we get the following characterization of $PsMAT^\lambda$:

Theorem 2. $PsPE_*(unit) = PsMAT^\lambda$

Due to the construction in Lemma 1 we not only have obtained a characterization of $PsMAT^\lambda$ by P systems with unit rules and energy assigned to membranes but also a normal form for this kind of P systems, i.e., only one symbol moving through a membrane structure is already sufficient (which of course is the minimal resource needed to obtain reasonable results).

The results obtained in this paper are already optimal with respect to the size of the multisets transported through a membrane, as in all proofs we needed only one object to be present in the system. Yet the optimal numbers of membranes necessary for obtaining computational completeness or for characterizing $PsMAT^\lambda$ still remain open problems (although we conjecture that the number of membranes needed in the universality results is already optimal). Some depictive examples and a few more details of the results presented in this paper can be found in [5].

Acknowledgements

This paper was inspired by a research proposal (see [10] for some details) exposed during the Second Brainstorming Week taking place in Sevilla in the first week of February, 2004. All authors acknowledge IST-2001-32008 project "MolCoNet".

References

1. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, Berlin (1989)
2. R. Freund: Generalized P-systems. In: G. Ciobanu, Gh. Păun (Eds.): Proceedings Fundamentals of Computation Theory. *Lecture Notes in Computer Science* **1684**. Springer-Verlag, Berlin (1999) 281–292
3. R. Freund: Sequential P-systems. *Romanian Journal of Information Science and Technology* **4**, 1-2 (2001) 77–88
4. R. Freund: Energy-controlled P systems. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): *Membrane Computing*. International Workshop, WMC-CdeA 2002, Curtea de Argeş, Romania, August 2002. *Lecture Notes in Computer Science* **2597**. Springer-Verlag, Berlin (2003) 247–260
5. R. Freund, A. Leporati, M. Oswald, C. Zandron: Sequential P systems with unit rules and energy assigned to membranes. In: [15] (2004) 168–182
6. R. Freund, M. Oswald: GP Systems with forbidding context. *Fundamenta Informaticae* **49**, 1-3 (2002) 81–102
7. R. Freund, M. Oswald: P systems with conditional communication rules assigned to membranes. To appear in *JALC*
8. R. Freund, Gh. Păun: On the number of non-terminals in graph-controlled, programmed, and matrix grammars. In: M. Margenstern, Y. Rogozhin (Eds.): Proc. Conf. Universal Machines and Computations, Chişinău (2001). *Lecture Notes in Computer Science* **2055**. Springer-Verlag, Berlin (2001) 214–225
9. R. Freund, Gh. Păun: From regulated rewriting to computing with membranes: collapsing hierarchies. *Theoretical Computer Science* **312** (2004) 143–188

10. A. Leporati, G. Mauri, C. Zandron: Simulating the Fredkin gate with energy–based P systems. In: [15] (2004) 292–308
11. M. L. Minsky: *Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey (1967)
12. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences* **61**, 1 (2000) 108–143 and TUCS Research Report 208 (1998) (`http://www.tucs.fi`)
13. Gh. Păun: *Membrane Computing: an Introduction*. Springer-Verlag, Berlin (2002)
14. Gh. Păun, Y. Suzuki, H. Tanaka: P Systems with energy accounting. *Int. J. Computer Math.* **78**, 3 (2001) 343–364
15. Gh. Păun, A. Riscos Nuñez, A. Romero Jiménez, F. Sancho Caparrini (Eds.): *Second Week on Membrane Computing*. Sevilla, Spain, Feb. 2-7, 2004. Dept. of Computer Sciences and Artificial Intelligence, Univ. of Sevilla Tech. Report 01/2004 (2004)
16. Rozenberg, G., Salomaa, A. (Eds.): *Handbook of Formal Languages*. Springer-Verlag, Berlin, Heidelberg (1997)
17. The P Systems Web Page, `http://psystems.disco.unimib.it`

Hierarchies of DLOGTIME-Uniform Circuits

Chuzo Iwamoto[1], Naoki Hatayama[1], Kenichi Morita[1], Katsunobu Imai[1], and Daisuke Wakamatsu[2]

[1] Hiroshima University, Graduate School of Engineering
Higashi-Hiroshima, 739-8527 Japan
chuzo@hiroshima-u.ac.jp
[2] Murata Machinery, Kyoto 612-8418, Japan

Abstract. We present complexity hierarchies on circuits under two DLOGTIME-uniformity conditions. It is shown that there is a language which can be recognized by a family of U_{E}-uniform circuits of depth $d(1+\epsilon)(\log n)^{r_1}$ and size $n^{r_2(1+\epsilon)}$ but not by any family of U_{E}-uniform circuits of depth $d(\log n)^{r_1}$ and size n^{r_2}, where $\epsilon > 0$, $d > 0$, $r_1 > 1$, and $r_2 \geq 1$ are arbitrary rational constants. It is also shown that there is a language which can be recognized by a family of U_{D}-uniform circuits of depth $(1+o(1))t(n)\log z(n)$ and size $(16t(n)+\psi(n)(\log z(n))^2)(z(n))^2$ but not by any family of U_{D}-uniform circuits of depth $t(n)$ and size $z(n)$, where $\psi(n)$ is an arbitrary slowly growing function not bounded by $O(1)$.

1 Introduction

One of the basic problems in complexity theory is to find the slightest enlarging of the complexity bound which allows new functions to be computed. There is a huge amount of literature on hierarchy results on various models, such as Turing machines [2,5,7,8,9,11,18,21,22], random access machines [4,12,17], and cellular automata [14,16]. In this paper, we investigate parallel complexity hierarchies based on uniform families of circuits.

For circuits, there are four major uniformities, $U_{\mathrm{B}}, U_{\mathrm{BC}}, U_{\mathrm{D}}$, and U_{E}, in increasing order of strength [19]. It is well known that presenting a proper hierarchy of parallel complexity classes is very difficult. For example, it is not known that NC^k is strictly included in NC^{k+1}, where NC^k is the class of languages accepted by U_{BC}-uniform (= logspace-uniform) circuits of depth $O((\log n)^k)$ and size polynomial. Furthermore, it is not even known whether $\mathrm{NC}^1 \overset{?}{\neq} \mathrm{NP}$ or all sets in the class P are accepted by U_{BC}-uniform circuits of linear size and logarithmic depth. Therefore, it is hopeless to try to present hierarchies for U_{BC}-uniform circuits.

Iwama and Iwamoto [12] presented a hierarchy result under U_{E}-uniformity, which is the strongest uniformity among the above four uniformities. (U_{E}-uniform (resp. U_{D}-uniform) is also called DLOGTIME-uniform using the Extended (resp. Direct) connection language.) It was shown that there are constants c_1, c_2, and a language L such that L can be recognized by a family of U_{E}-uniform circuits of depth $c_1 t(n)$ and size $(z(n))^{c_2}$ but not by any family of

U_{E}-uniform circuits of depth $t(n)$ and size $z(n)$, where $t(n) \neq O(\log z(n))$ is a polynomial in $\log z(n)$.

In this paper, we tighten this hierarchy result. It is shown that there is a language which can be recognized by a family of U_{E}-uniform circuits of depth $d(1+\epsilon)(\log n)^{r_1}$ and size $n^{r_2(1+\epsilon)}$ but not by any family of U_{E}-uniform circuits of depth $d(\log n)^{r_1}$ and size n^{r_2}, where $\epsilon > 0$, $d > 0$, $r_1 > 1$, and $r_2 \geq 1$ are arbitrary rational constants. Namely, constants c_1 and c_2 in [12] can be replaced by an arbitrarily small constant $1+\epsilon$ when $t(n) = d(\log n)^{r_1}$ and $z(n) = n^{r_2}$.

Furthermore, we present a hierarchy result under U_{D}-uniformity. It is shown that there is a language which can be recognized by a family of U_{D}-uniform circuits of depth $(1+o(1))t(n)\log z(n)$ and size $(16t(n)+\psi(n)(\log z(n))^2)(z(n))^2$ but not by any family of U_{D}-uniform circuits of depth $t(n)$ and size $z(n)$, where $\psi(n)$ is an arbitrary slowly growing function not bounded by $O(1)$. The hierarchy in [12] uses the so-called depth-universal circuit [3], whose size is not so efficient; in this paper, we construct a new universal circuit which is suitable for U_{D}-uniformity.

Since we consider fan-in 2 circuits in this paper, our hierarchy results do not hold for depth less than $\log n$. In the class NC^1, several separation results have been known. For example, there is a noncollapsing hierarchy in AC^0, which is the class of languages recognized by constant depth, polynomial-size, unbounded fan-in circuits. It is known [20] that there are languages in $\mathrm{AC}^0_k - \mathrm{AC}^0_{k-1}$ for each $k > 0$, where AC^0_k is the class of languages recognized by DLOGTIME-uniform, depth-k, polynomial-size, unbounded fan-in circuits. Also, it is known [1,6] that the exclusive OR function is not in AC^0, which implies that $\mathrm{AC}^0 \subsetneq \mathrm{NC}^1$.

For other parallel models, several papers presented proper hierarchies. It was shown that there is a constant d such that $dt(n)$-time PRAMs with $p(n)$ processors are more powerful than $t(n)$-time PRAMs with the same $p(n)$ processors [12]. It was also shown that $t_2(n)$-time $s(n)$-space parallel TMs with $p(n)$ processors are more powerful than $t_1(n)$-time $s(n)$-space parallel TMs with $p(n)$ processors if $t_2(n) \neq O(t_1(n)\log s(n))$ [15].

In Section 2, we give definitions of circuits and uniformities. The main theorems are also given in that section. The proofs are given in Sections 3 and 4.

2 Definitions and Results

The definitions of circuits are mostly from [19]. A *combinatorial circuit* is a directed acyclic graph, where each node (gate) has indegree $d \leq 2$ and is labeled by some Boolean function of d variables. Nodes with indegree 0 (resp. outdegree 0) are inputs (resp. outputs). In this paper, we consider a *family* of circuits $C = (\alpha_1, \alpha_2, \ldots, \alpha_n, \ldots)$, where α_n has n inputs and one output. We assume that gate 0 is the output and gates $1, \ldots, n$ are the inputs. We denote the depth and size of α_n by $t(n)$ and $z(n)$, respectively.

Let $gate(g, p)$ denote the gate reached by following the path $p \in \{L, R\}^*$ towards the inputs of a circuit. For example, $gate(g, \varepsilon)$ is gate g, $gate(g, L)$ is gate g's left input, $gate(g, LR)$ is gate g's left input's right input, and so on.

The *standard encoding* $\overline{\alpha}_n$ of a circuit α_n is a string of 4-tuples $\langle n, g, p, y \rangle$, where $g \in \{0,1\}^*$, $p \in \{\varepsilon, L, R\}$, and $y \in \{\wedge, \vee, \neg\} \cup \{0,1\}^*$, such that in α_n either (i) $p = \varepsilon$ and gate g is a y-gate, $y \in \{\wedge, \vee, \neg\}$, or (ii) $p \neq \varepsilon$ and $gate(g,p)$ is gate y, $y \in \{0,1\}^*$. The *direct connection language* $L_{\mathrm{DC}}(C)$ of a circuit family C is the set of strings of the form $\langle n, g, p, y \rangle$. The *extended connection language* $L_{\mathrm{EC}}(C)$ is as above, except $p \in \{L, R\}^*$ and $|p| \leq \log z(n)$.

A family of circuits $C = (\alpha_1, \alpha_2, \ldots, \alpha_n, \ldots)$ of size $z(n)$ is said to be U_{BC}-*uniform* if the mapping $n \to \overline{\alpha}_n$ is computable by an $O(\log z(n))$-space deterministic TM. A family of circuits of size $z(n)$ is said to be U_{D}-*uniform* (resp. U_{E}-*uniform*) if there is an $O(\log z(n))$-time deterministic TM recognizing $L_{\mathrm{DC}}(C)$ (resp. $L_{\mathrm{EC}}(C)$). Now we are ready to present our main theorems.

Theorem 1. *Let $t(n) = d(\log n)^{r_1}$ and $z(n) = n^{r_2}$, where $d > 0$, $r_1 > 1$, and $r_2 \geq 1$ are arbitrary rational constants. For any small rational constant $\epsilon > 0$, there is a language which can be recognized by a family of U_{E}-uniform circuits of depth $(1+\epsilon)t(n)$ and size $(z(n))^{1+\epsilon}$ but not by any family of U_{E}-uniform circuits of depth $t(n)$ and size $z(n)$.*

The proof of this theorem is given in Section 3. It was shown that there are constants c_1, c_2, and a language L such that L can be recognized by a family of U_{E}-uniform circuits of depth $c_1 t(n)$ and size $(z(n))^{c_2}$ but not by any family of U_{E}-uniform circuits of depth $t(n)$ and size $z(n)$ [12]. Theorem 1 improves both the constant factor c_1 of depth and the exponent c_2 of size to $1+\epsilon$ simultaneously. Although the values of c_1 and c_2 were not mentioned in [12], a careful analysis shows $c_1 = c_2 = 3+o(1)$. The next theorem is a hierarchy of U_{D}-uniform circuits.

Theorem 2. *Suppose that $z(n) \geq n$, $t(n) \geq \log z(n)$, and $\psi(n) \neq O(1)$ are arbitrary functions such that $\lceil \log z(n) \rceil$, $t(n)$, and $\psi(n)$ are computable by $O(\log z(n))$-time deterministic TMs if input n is given as a binary string of length $\lfloor \log n \rfloor + 1$. There is a language which can be recognized by a family of U_{D}-uniform circuits of depth $(1 + o(1))t(n) \log z(n)$ and size $(16t(n) + \psi(n)(\log z(n))^2)(z(n))^2$ but not by any family of U_{D}-uniform circuits of depth $t(n)$ and size $z(n)$.*

The proof is given in Section 4. It is known [12] that functions computable by $O(\log z(n))$-time TMs include most common polylogarithmic functions, such as $\log n \log\log n$ and $c(\log n)^k$.

3 Hierarchy of U_{E}-Uniform Circuit Families

In this section, we prove Theorem 1. Let $U_{\mathrm{E}}(t(n), z(n))$ denote the class of languages recognized by families of U_{E}-uniform circuits of depth $t(n)$ and size $z(n)$.

3.1 Translation Lemma for U_{E}-Uniform Circuit Families

Lemma 1. *Suppose that $t_1(n), t_2(n) \geq \log n$ and $z_1(n), z_2(n) \geq n$ are arbitrary polylogarithmic and polynomial functions, respectively, such that $t_1(n)$, $t_2(n)$,*

$\log z_1(n)$, and $\log z_2(n)$ are computable by $O(\log n)$-time deterministic TMs if input n is given as a binary string of length $\lfloor \log n \rfloor + 1$. If $U_{\rm E}(t_1(n)+1, z_1(n)+n) \subseteq U_{\rm E}(t_2(n), z_2(n))$, then $U_{\rm E}(t_1(2^{kn}), z_1(2^{kn})) \subseteq U_{\rm E}(t_2(2^{kn}), z_2(2^{kn}))$, where $k \geq 1$ is an arbitrary integer.

The set of functions computable by $O(\log n)$-time deterministic TMs includes $d(\log n)^{r_1}$ and $r_2 \log n$. In the rest of this subsection, we prove Lemma 1.

Let $C_1 = (\alpha_1, \alpha_2, \ldots, \alpha_n, \ldots)$ be a family of $U_{\rm E}$-uniform circuits of depth $t_1(2^{kn})$ and size $z_1(2^{kn})$. Let L_1 be the language recognized by C_1. We define a language L'_1 as $L'_1 = \{x1^l \mid x \in L_1 \text{ and } |x| + l = 2^{k|x|}\}$, where 1^l is a padding sequence of length l.

We construct a family of $U_{\rm E}$-uniform circuits $C'_1 = (\alpha'_1, \alpha'_2, \ldots, \alpha'_n, \ldots)$ which recognizes the language L'_1 as follows. Consider the nth circuit α'_n. If n cannot be written as $n = 2^{ki}$ for any integer i, then α'_n accepts the empty set. If $n = 2^{ki}$, then α'_n is composed of the ith circuit α_i of C_1, an $(n-i)$-input AND-circuit (of depth $\log(n-i)$ and size $n-i-1$), and an AND-gate. The inputs of the AND-gate are the outputs of α_i and the $(n-i)$-input AND-circuit. Therefore, the depth and size of C'_1 are bounded by $t_1(2^{ki}) + 1$ $(= t_1(n) + 1)$ and $z_1(2^{ki}) + n - i$ $(\leq z_1(n) + n)$, respectively.

Recall that the assumption of Lemma 1 is $U_{\rm E}(t_1(n) + 1, z_1(n) + n) \subseteq U_{\rm E}(t_2(n), z_2(n))$. From this assumption and the previous paragraph, one can see that there is a family of $U_{\rm E}$-uniform circuits $C'_2 = (\beta'_1, \beta'_2, \ldots, \beta'_n, \ldots)$ of depth $t_2(n)$ and size $z_2(n)$ which recognizes the language L'_1.

We construct a family of $U_{\rm E}$-uniform circuits $C_2 = (\beta_1, \beta_2, \ldots, \beta_n, \ldots)$ which recognizes the language L_1 as follows. Consider the 2^{kn}th circuit $\beta'_{2^{kn}}$ of C'_2. By changing the last $l = 2^{kn} - n$ inputs of $\beta'_{2^{kn}}$ into constant value 1, we obtain a circuit of n inputs. This is the circuit β_n. Note that $\beta'_{2^{kn}}$ has depth $t_2(2^{kn})$ and size $z_2(2^{kn})$, and both $\beta'_{2^{kn}}$ and β_n have the same structure. Hence, β_n also has depth $t_2(2^{kn})$ and size $z_2(2^{kn})$. This completes the proof of Lemma 1.

3.2 Proof of Theorem 1

It remains to show Theorem 1 by using Lemma 1 and a hierarchy result in [12]. The proof is similar to [10], but new techniques are introduced in order to tighten both the constant factor of depth and the exponent of size simultaneously.

Let p and q are sufficiently large integers satisfying $r_2 = p/q$, $(1 + 1/p)^{r_1} < 1 + \epsilon$, and $1/q < \epsilon$. Then, the following proposition holds.

Proposition 1. *There is an integer n_0 such that $n^{(p+1)/q} + n \leq n^{r_2(1+\epsilon)}$ for all $n \geq n_0$.*

Proof. Since $r_2 = p/q$ and $r_2 + \epsilon \leq r_2(1+\epsilon)$, we prove that *there is an integer n_0 such that $n^{r_2+(1/q)} + n \leq n^{r_2+\epsilon}$ for all $n \geq n_0$*. Assume for contradiction that, for any (large) integer n'_0, there is an integer n' such that $n' \geq n'_0$ and

$$(n')^{r_2+(1/q)} + (n') > (n')^{r_2+\epsilon}.$$

Dividing both sides of this inequality by $(n')^{r_2+(1/q)}$ yields

$$1 + \frac{1}{(n')^{(r_2-1)+(1/q)}} > (n')^{\epsilon-(1/q)}.$$

Recall that $r_2 \geq 1$ and $\epsilon > 1/q$. If we let $n'_0 \to \infty$, then the left-hand side asymptotically becomes 1 while the right-hand side becomes infinitely large, a contradiction. ∎

From Proposition 1, there are integers p, q, and n_0 such that

$$n^{r_2} = n^{p/q} < n^{(p+1)/q} + n \leq n^{r_2(1+\epsilon)}$$

for all $n \geq n_0$. Let $d' = \frac{d}{(p/q)^{r_1}}$. Then, the following inequality holds:

$$d'((p+1)/q)^{r_1} = \frac{d}{(p/q)^{r_1}}((p+1)/q)^{r_1} = d(1+1/p)^{r_1} < d(1+\epsilon)$$

Furthermore, if we consider sufficiently large n such that

$$(d(1+\epsilon) - d'((p+1)/q)^{r_1})(\log n)^{r_1} \geq 1,$$

then

$$d'((p+1)/q)^{r_1}(\log n)^{r_1} + 1 \leq d(1+\epsilon)(\log n)^{r_1}.$$

Therefore, to show Theorem 1, it is enough to prove the following relation (1):

$$U_{\mathrm{E}}(d'(p/q)^{r_1}(\log n)^{r_1}, n^{p/q}) \subsetneq U_{\mathrm{E}}(d'((p+1)/q)^{r_1}(\log n)^{r_1} + 1, n^{(p+1)/q} + n) \quad (1)$$

Assume for contradiction that the relation (1) does not hold; namely,

$$U_{\mathrm{E}}(d'((p+1)/q)^{r_1}(\log n)^{r_1} + 1, n^{(p+1)/q} + n) \subseteq U_{\mathrm{E}}(d'(p/q)^{r_1}(\log n)^{r_1}, n^{p/q}).$$

From Lemma 1, we obtain

$$U_{\mathrm{E}}(d'(k(p+1)n/q)^{r_1}, 2^{k(p+1)n/q}) \subseteq U_{\mathrm{E}}(d'(kpn/q)^{r_1}, 2^{kpn/q}). \quad (2)$$

Let i be an integer. By substituting $k = (p+i)q$ into the relation (2), we obtain

$$U_{\mathrm{E}}(d'((p+1)(p+i)n)^{r_1}, 2^{(p+1)(p+i)n}) \subseteq U_{\mathrm{E}}(d'(p(p+i)n)^{r_1}, 2^{p(p+i)n}). \quad (3)$$

When $i \geq 1$, $p(p+i) \leq (p+1)(p+i-1)$ holds. Thus, if $i \geq 1$, then relation (3) can be rewritten as

$$\begin{aligned} &U_{\mathrm{E}}(d'((p+1)(p+i)n)^{r_1}, 2^{(p+1)(p+i)n}) \\ &\qquad \subseteq U_{\mathrm{E}}(d'((p+1)(p+i-1)n)^{r_1}, 2^{(p+1)(p+i-1)n}). \end{aligned} \quad (4)$$

Substituting $i = 0$ into (3) yields

$$U_{\mathrm{E}}(d'((p+1)pn)^{r_1}, 2^{(p+1)pn}) \subseteq U_{\mathrm{E}}(d'(p^2n)^{r_1}, 2^{p^2n}). \quad (5)$$

Let c be an integer (the value of c will be fixed later). Substituting $i = 1, 2, \cdots, (c-1)p$ into (4) yields

$$\left.\begin{aligned}
&U_{\mathrm{E}}(d'((p+1)(p+1)n)^{r_1}, 2^{(p+1)(p+1)n}) \\
&\qquad\qquad \subseteq U_{\mathrm{E}}(d'((p+1)pn)^{r_1}, 2^{(p+1)pn}), \\
&U_{\mathrm{E}}(d'((p+1)(p+2)n)^{r_1}, 2^{(p+1)(p+2)n}) \\
&\qquad\qquad \subseteq U_{\mathrm{E}}(d'((p+1)(p+1)n)^{r_1}, 2^{(p+1)(p+1)n}), \\
&\qquad\qquad \vdots \\
&U_{\mathrm{E}}(d'((p+1)(cp)n)^{r_1}, 2^{(p+1)(cp)n}) \\
&\qquad\qquad \subseteq U_{\mathrm{E}}(d'((p+1)(cp-1)n)^{r_1}, 2^{(p+1)(cp-1)n}).
\end{aligned}\right\} \tag{6}$$

Since $cp^2n < (p+1)(cp)n$ and $r_1 > 1$, it is clear that

$$U_{\mathrm{E}}(cd'(p^2n)^{r_1}, 2^{cp^2n}) \subseteq U_{\mathrm{E}}(d'((p+1)(cp)n)^{r_1}, 2^{(p+1)(cp)n}). \tag{7}$$

From inclusion relations (5), (6), and (7), we obtain

$$U_{\mathrm{E}}(cd'(p^2n)^{r_1}, 2^{cp^2n}) \subseteq U_{\mathrm{E}}(d'(p^2n)^{r_1}, 2^{p^2n}). \tag{8}$$

On the other hand, it is known [12] that there are constants c_1 and c_2 such that

$$U_{\mathrm{E}}(t(n), z(n)) \subsetneq U_{\mathrm{E}}(c_1t(n), (z(n))^{c_2}). \tag{9}$$

If we choose c as an integer such that $c \geq c_1$ and $c \geq c_2$, then the relation (8) contradicts (9). This completes the proof of Theorem 1.

Remark 1. In [12], $z(n) \geq n$ and $t(n) \neq O(\log n)$ were defined as polynomial and polylogarithmic functions, respectively, such that $\log z(n)$ and $t(n)$ are computable by $O(\log n)$-time TMs. However, the hierarchy in [12] holds also for all functions $z(n) \geq n$ and $t(n) \neq O(\log z(n))$ such that $\log z(n)$ and $t(n)$ are computable by $O(\log z(n))$-time TMs.

4 Hierarchy of U_{D}-Uniform Circuit Families

We construct a family of circuits α_n of depth $(1 + o(1))t(n) \log z(n)$ and size $(16t(n) + \psi(n)(\log z(n))^2)(z(n))^2$ such that α_n can recognize a language which cannot be recognized by any family of circuits β_n of depth $t(n)$ and size $z(n)$.

4.1 High-Level Description

The circuit α_n is composed of four subcircuits, $\alpha_n^{\mathrm{tm}}, \alpha_n^{\mathrm{type}}, \alpha_n^{\mathrm{con}}$ and α_n^{sim} (see Fig. 1). The output of α_n^{tm} is 1 iff there is a TM, say, T_b, such that the input string b is an encoding of T_b. Let L_{DC} be the direct connection language accepted by T_b, and let β_n be the circuit defined by L_{DC}. Circuit α_n^{type} computes the type of each gate of β_n by simulating T_b. Similarly, α_n^{con} computes the connection between β_n's gates by simulating T_b. Circuit α_n^{sim} simulates β_n on input string b,

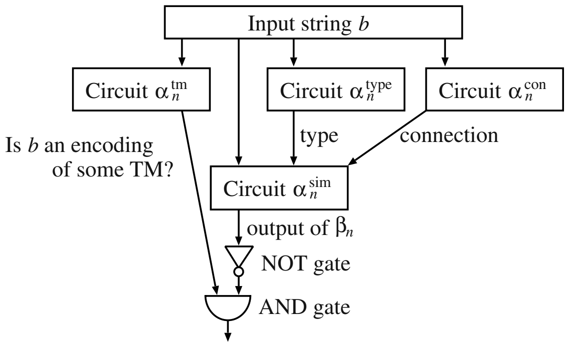

Input string b
Circuit α_n^{tm}
Circuit α_n^{type}
Circuit α_n^{con}
Is b an encoding of some TM?
type
connection
Circuit α_n^{sim}
output of β_n
NOT gate
AND gate

Fig. 1. High-level description of α_n

i.e., α_n^{sim} outputs 1 iff β_n outputs 1. The output of α_n is 1 iff the outputs of α_n^{tm} and α_n^{sim} are 1 and 0, respectively. In Lemma 2, we will show that the depth and size of α_n are $(1 + o(1))t(n) \log z(n)$ and $(16t(n) + \psi(n)(\log z(n))^2)(z(n))^2$, respectively. In Lemma 3, we will show that the language, recognized by a family of U_{D}-uniform circuits α_n, cannot be recognized by any family of U_{D}-uniform circuits of depth $t(n)$ and size $z(n)$.

It is known [12] that there is a family of U_{E}-uniform circuits of depth $O(\log n)$ and size $\le n^2$ which can decide whether a given string is an encoding of some TM. Thus, we omit the description of α_n^{tm}. If the given string b is not an encoding, then α_n^{tm} outputs 0, and thus the output of α_n becomes 0, regardless of the outputs of α_n^{type}, α_n^{con} and α_n^{sim}. Therefore, in the following, we assume that the input b is an encoding of some TM T_b.

4.2 Circuits $\boldsymbol{\alpha_n^{\mathrm{sim}}}$

In [12], the depth-universal circuit [3] was used for α_n^{sim}. In this paper, we construct a new universal circuit whose size is smaller than the depth-universal circuit constructed in [3]. The idea is illustrated in Fig. 2 (the basic idea was presented by the same author at [13].)

Let $z'(n) = 2^{\lceil \log z(n) \rceil}$. Since we assumed that $\lceil \log z(n) \rceil$ is computable by an $O(\log z(n))$-time TM, the value $z'(n)$ is also computable by an $O(\log z(n))$-time TM. Note that $z'(n) < 2z(n)$. Circuit α_n^{sim} has subcircuits

$$c_{\mathrm{gate}}(t, n+1), c_{\mathrm{gate}}(t, n+2), \ldots, c_{\mathrm{gate}}(t, z'(n))$$

for $1 \le t \le t(n)$, which are represented by $\boxed{n+1}$ through $\boxed{z'(n)}$ in Fig. 2(a). Each circuit $c_{\mathrm{gate}}(t, i)$ corresponds to the gate g_i of β_n. Each $c_{\mathrm{gate}}(t, i)$ is composed of eight gates shown in Fig. 2(c). Each $c_{\mathrm{gate}}(t, i)$ receives three inputs from the left side, which determine the type of g_i. These three inputs are given by the circuit α_n^{type}. Circuit $c_{\mathrm{gate}}(0, 0)$, represented by $\boxed{0}$, corresponds to the output gate of β_n and is also composed of the eight gates in Fig. 2(c). Circuits

$$c_{\mathrm{gate}}(0, 1), c_{\mathrm{gate}}(0, 2), \ldots, c_{\mathrm{gate}}(0, n),$$

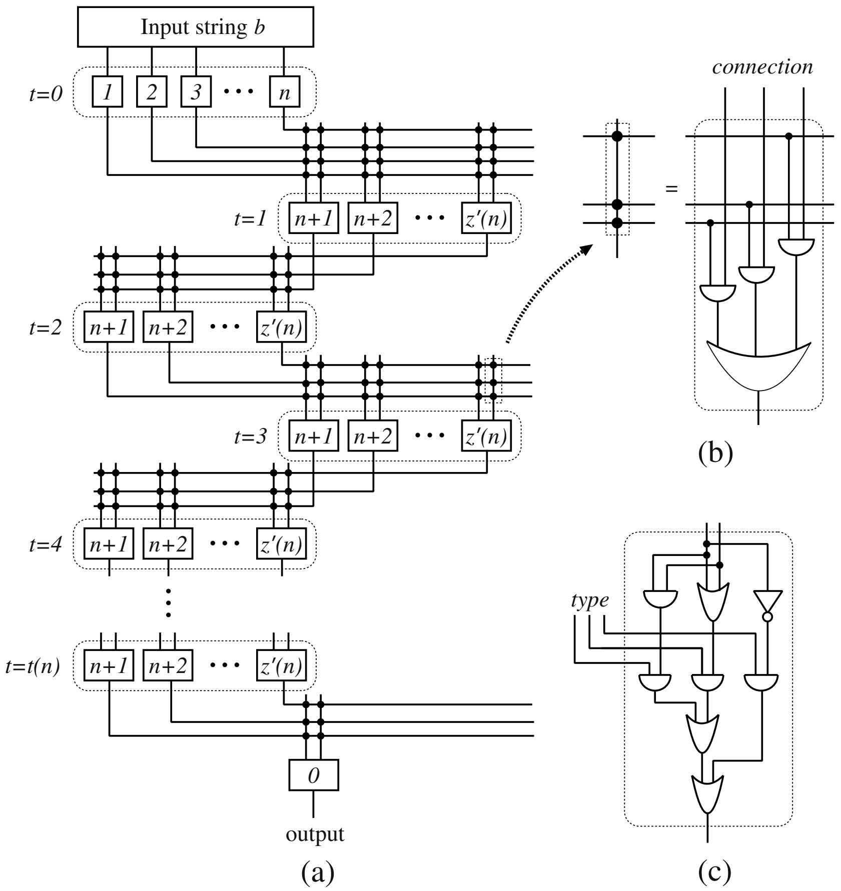


Fig. 2. (a) Circuit α_n^{sim}, (b) Connection between gates, (c) Circuit $c_{\text{gate}}(t, i)$

represented by $\boxed{1}$ through $\boxed{n}$, are β_n's input gates which are AND-gates of fan-in 1.

A set of grids in Fig. 2(b) is composed of $(z'(n) - n)$ AND-gates of fan-in 2 and an OR-gate of fan-in $z'(n) - n$ (= a fan-in-2 circuit of depth $\log(z'(n) - n)$ and size $z'(n) - n - 1$). The inputs of the AND-gates in Fig. 2(b) determine the connection between t and $t + 1$. Those are given by the circuit α_n^{con}.

Each gate can be uniquely encoded to a string of length $O(\log z(n))$ in binary. It is not hard to see that the direct connection language of α_n^{sim} can be recognized by an $O(\log z(n))$-time TM.

4.3 Circuits α_n^{type} and α_n^{con}

Recall that L_{DC} is the direct connection language accepted by T_b, and β_n is the circuit defined by L_{DC}. In order to find the type of each gate g of β_n, we decide whether $\langle n, g, \varepsilon, y\rangle$ is in L_{DC} for all $y \in \{\wedge, \vee, \neg\}$ and all gates g by simulating T_b. Thus, α_n^{type} has subcircuits $c_n^{\mathrm{type}}(g, y)$ which check whether $\langle n, g, \varepsilon, y\rangle$ is in L_{DC}, i.e., the type of gate g is y.

Let $\psi(n) \neq O(1)$ be an arbitrary slowly growing function computable by an $O(\log z(n))$-time TM. We consider k-tape k-state TM T_b which uses 0 and 1 as its tape symbols. We do not know the value k previously, so we construct subcircuits $c_n^{\mathrm{type}}(g, y)$ for all $1 \le k \le \psi'(n)$, where $\psi'(n)$ be an arbitrary $O(\log z(n))$-time-computable function such that $\psi'(n) \neq O(1)$ and $(\psi'(n))^6 2^{\psi'(n)} \le \psi(n)$. (For example, let $\psi'(n) = \log\log\psi(n)$. The reason why we need such a $\psi'(n)$ is given in Section 4.4.) In Lemma 3, we will consider a sufficiently long input string b such that $k \le \psi'(|b|)$. (Strictly speaking, $c_n^{\mathrm{type}}(g, y)$ should be written as $c_n^{\mathrm{type}}(g, y, k)$. For simplicity of notation, we omit k.) The structure of $c_n^{\mathrm{type}}(g, y)$ is similar to [12], but circuits constructed in this paper are much smaller.

We represent the configuration of T_b at step t by four words

$$s(g, y; t), w_0(g, y; t, i, j), w_1(g, y; t, i, j), h(g, y; t, i, j),$$

where $s(g, y; t)$ is a $\log k$-bit word, and the remaining three words are single bits. $s(g, y; t)$ represents the state of T_b at step t. If the jth cell of the ith tape of T_b contains symbol 0 (resp. 1) at step t, then $w_0(g, y; t, i, j) = 1$ (resp. $w_1(g, y; t, i, j) = 1$); otherwise $w_0(g, y; t, i, j) = 0$ (resp. $w_1(g, y; t, i, j) = 0$). If the head of the ith tape of T_b is placed at the jth cell at step t, then $h(g, y; t, i, j) = 1$; otherwise $h(g, y; t, i, j) = 0$.

Suppose that the transition rules of T_b are numbered $1, 2, \ldots, f, \ldots, k2^k$. We transform each rule f by the following words

$$p(f), a(f; i), q(f), b(f; i), d(f; i),$$

where $p(f)$ and $q(f)$ are $\log k$-bit words, and $a(f; i), b(f; i)$, and $d(f; i)$ are single bits. Those words mean that if the state is $p(f)$ and the ith head is reading the symbol $a(f; i)$ for each i, then the state is changed into $q(f)$ and the ith head writes $b(f; i)$ and moves to the right (left) if $d(f; i) = 1$ ($d(f; i) = 0$). This transformation can be done by a circuit whose depth is approximately $\log k$.

Now we show how to simulate a single step of TM T_b. For $t = 0$, $s(g, y; 0)$ represents the initial state, $w_0(g, y; 0, 1, j)$ and $w_1(g, y; 0, 1, j)$ for $j \ge 1$ contain the input string $\langle n, g, \varepsilon, y\rangle$, and $h(g, y; 0, i, 1) = 1$ for $1 \le i \le k$. The remaining words $w_0(g, y; 0, i, j), w_1(g, y; 0, i, j)$, and $h(g, y; 0, i, j)$ are $1, 0$, and 0, respectively. In the following, we show the connection between steps t and $t + 1$.

We can decide whether two l-bit words y, z are the same by $EQ(y, z) = \bigwedge_{i=1}^{l} \big(y_i z_i \vee (\neg y_i)(\neg z_i)\big)$, where y_i and z_i are the ith bit of y and z, respectively. We compare the current state and $p(f)$ by

$$\textit{cmp-state}(g, y, f; t) = EQ\big(s(g, y; t), p(f)\big).$$

Since $s(g, y; t)$ and $p(f)$ are $\log k$-bit words, the depth is approximately $\log\log k$. We then compare the symbol read by the ith head of T_b and $a(f; i)$ by

$$\textit{cmp-sybl}_1(g, y, f; t, i) = \bigvee_{j=1}^{\psi'(n)\log z'(n)} \big(w_1(g, y; t, i, j) \wedge a(f; i) \wedge h(g, y; t, i, j)\big),$$

$$cmp\text{-}sybl_0(g, y, f; t, i) = \bigvee_{j=1}^{\psi'(n) \log z'(n)} \bigl(w_0(g, y; t, i, j) \wedge \neg a(f; i) \wedge h(g, y; t, i, j)\bigr).$$

Here, we must consider the leftmost $\psi'(n) \log z'(n)$ cells on each tape for $\psi'(n) \neq O(1)$, since T_b is an $O(\log z(n))$-time TM. An OR-gate of fan-in $\psi'(n) \log z'(n)$ can be replaced by a circuit of depth $\log \log z'(n) + \log \psi'(n)$. We define *cmp-sybl*$(g, y, f; t, i)$ as

$$cmp\text{-}sybl(g, y, f; t, i) = cmp\text{-}sybl_1(g, y, f; t, i) \vee cmp\text{-}sybl_0(g, y, f; t, i).$$

Then *cmp-sybl*$(g, y, f; t, i) = 1$ iff the ith head of T_b is reading $a(f; i)$. Therefore, the current configuration agrees with the transition rule f iff the following *agree*$(g, y, f; t)$ is 1.

$$agree(g, y, f; t) = cmp\text{-}state(g, y, f; t) \wedge \bigwedge_{i=1}^{k} cmp\text{-}sybl(g, y, f; t, i)$$

Now the next state can be computed by

$$s(g, y; t+1) = \bigvee_{f=1}^{k2^k} \bigl(q(f) \wedge agree(g, y, f; t)\bigr).$$

Tape symbol $w_1(g, y; t+1, i, j)$ is set to be $b(f; i)$ if the ith head is placed at the jth cell and writes 1 into the jth cell and the current configuration agrees with the transition rule f. Thus, tape symbols are updated as follows:

$$\begin{aligned} w_1(g, y; t+1, i, j) = {} & \bigl(w_1(g, y; t, i, j) \wedge \neg heads(g, y; t, i, j)\bigr) \vee \\ & \bigvee_{f=1}^{k2^k} \bigl(b(f; i) \wedge heads(g, y; t, i, j) \wedge agree(g, y, f; t)\bigr) \\ w_0(g, y; t+1, i, j) = {} & \bigl(w_0(g, y; t, i, j) \wedge \neg heads(g, y; t, i, j)\bigr) \vee \\ & \bigvee_{f=1}^{k2^k} \bigl(\neg b(f; i) \wedge heads(g, y; t, i, j) \wedge agree(g, y, f; t)\bigr) \end{aligned}$$

The head positions are updated if the head positions at step $t+1$ are adjacent to the positions at step t. We define $h(g, y; t+1, i, j)$ as

$$\begin{aligned} h(g, y; t+1, i, j) = \bigvee_{f=1}^{k2^k} \Bigl(& \bigl(h(g, y; t, i, j-1) \wedge d(f; i) \wedge agree(g, y, f; t)\bigr) \\ & \vee \bigl(h(g, y; t, i, j+1) \wedge \neg d(f; i) \wedge agree(g, y, f; t)\bigr)\Bigr). \end{aligned}$$

We omit the description of circuit α_n^{con}, since it is similar to α_n^{type}. The difference is as follows. In order to find the inputs of each gate g, we must decide whether $\langle n, g, p, y\rangle$ is in L_{DC} for every pair of gates g and y, where $p \in \{L, R\}$. Thus, α_n^{con} has subcircuits $c_n^{\mathrm{con}}(g, p, y)$ checking whether $\langle n, g, p, y\rangle$ is in L_{DC}. For example, we use $h(g, p, y; t, i, j)$ where $p \in \{L, R\}$ and $g, y \in \{0, 1\}^*$, instead of $h(g, y; t, i, j)$ where $g \in \{0, 1\}^*$ and $y \in \{\wedge, \vee, \neg\}$.

4.4 Analysis of Depth and Size

Let us start with the size of α_n^{con}. The number of gates $h(g, p, y; t, i, j)$ in α_n^{con} is calculated as $(z'(n) \cdot 2 \cdot z'(n)) \cdot (\psi'(n) \log z'(n) \cdot \psi'(n) \cdot \psi'(n) \log z'(n)) \cdot \psi'(n)$ $(=2(z'(n))^2(\psi'(n))^4(\log z'(n))^2)$, where the last $\psi'(n)$ is for considering k-tape k-state TMs for $1 \le k \le \psi'(n)$. Each $h(g, p, y; t, i, j)$ is the output of an OR-gate of fan-in $k2^k$ (= a fan-in-2 circuit of depth $k + \log k$ and size $k2^k - 1$). Therefore, the size of α_n^{con} is bounded by $O((z(n))^2(\psi'(n))^5 2^{\psi'(n)}(\log z(n))^2)$, which is further bounded by $\psi(n)(z(n))^2(\log z(n))^2$ for large n because $(\psi'(n))^6 2^{\psi'(n)} \le \psi(n)$. (Note that the number of the remaining gates, such as $w_0(g, p, y; t, i, j)$, is the same as $h(g, p, y; t, i, j)$.) The following lemmas complete the proof of Theorem 2.

Lemma 2. *Circuit α_n has depth $(1 + o(1))t(n) \log z(n)$ and size $(16t(n) + \psi(n)(\log z(n))^2)(z(n))^2$.*

Proof. α_n^{sim} is composed of $t(n)$ levels (see Fig. 2(a)), each of which has depth $(1 + \log z'(n)) + 4$ (see Figs. 2(b) and 2(c)). Therefore, the depth of α_n^{sim} is $t(n)(\log z'(n) + 5)$, which is bounded by $(1 + o(1))t(n) \log z(n)$. α_n^{type} simulates an $O(\log z(n))$-step TM T_b, and a single step needs depth $O(\log \log z(n))$ (see *cmp-sybl*$_0$). Therefore, α_n^{type} has depth $O(\psi'(n) \log z(n) \log \log z(n))$ ($\ll$ $t(n) \log z(n)$). Hence, the depth of α_n is bounded by $(1 + o(1))t(n) \log z(n)$.

Each level of α_n^{sim} is composed of $(z'(n) - n)$ subcircuits, each of which has $2((z'(n) - n) + (z'(n) - n - 1)) + 8$ gates (see Figs. 2(b) and 2(c)). Therefore, each level of α_n^{sim} has at most $(z'(n) - n)(2((z'(n) - n) + (z'(n) - n - 1)) + 8)$ gates, which is less than $4(z'(n))^2 \le 16(z(n))^2$. Recall that α_n^{con} has at most $\psi(n)(z(n))^2(\log z(n))^2$ gates, and α_n^{tm} and α_n^{type} are much smaller than α_n^{con}. Therefore, the size of α_n is $(16t(n) + \psi(n)(\log z(n))^2)(z(n))^2$. ■

Lemma 3. *Any family of U_{D}-uniform circuits of depth $t(n)$ and size $z(n)$ cannot recognize the language which is recognized by the above-defined α_n.*

Proof. Assume for contradiction that there is a family of U_{D}-uniform circuits, say, β_n, of depth $t(n)$ and size $z(n)$ which can recognize the language recognized by α_n. Since β_n is U_{D}-uniform, there is an $O(\log z(n))$-time k-state k-tape TM T_b which recognizes the direct connection language L_{DC} of β_n for some constant k.

Consider a sufficiently long encoding string b of TM T_b such that $k \le \psi'(|b|)$. If such a string b is given to α_n as its input, then (i) α_n^{tm} outputs 1, (ii) α_n^{type} correctly outputs the type of every gate of β_n, (iii) α_n^{con} outputs the connection between every pair of β_n's gates, and therefore (iv) α_n^{sim} outputs 0 iff β_n outputs 0. Recall that α_n outputs 1 iff α_n^{tm} and α_n^{sim} output 1 and 0, respectively. Therefore, α_n outputs 1 iff β_n outputs 0, a contradiction. ■

References

1. M. Ajtai, Σ_1^1-formulae on finite structure, *Ann. Pure Appl. Logic*, **24** (1983) 1–48.
2. S.A. Cook, A hierarchy for nondeterministic time complexity, *J. Comput. System Sci.*, **7** (1973) 343–353.
3. S.A Cook and H.J. Hoover, "A depth-universal circuit," *SIAM J. Comput.*, **14**(4), 833–839, 1985.
4. S.A. Cook and R.A. Reckhow, Time bounded random access machines, *J. Comput. System Sci.*, **7** (1973) 354–375.
5. M. Fürer, The tight deterministic time hierarchy, *Proc. 14th Annual ACM Symp. on Theory of Computing*, San Francisco, California, 1982, 8–16.
6. M. Furst, J. Saxe, and M. Sipser, Parity, circuits, and the polynomial time hierarchy, *Math. Systems Theory*, **17** (1984) 12–27.
7. J. Hartmanis, P.M. Lewis II, and R.E. Stearns, Hierarchies of memory limited computations, *Proc. 6th Annual IEEE Symp. on Switching Circuit Theory and Logical Design*, 1965, 179–190.
8. J. Hartmanis and R.E. Stearns, On the computational complexity of algorithms, *Trans. Amer. Math. Soc.*, **117** (1965) 285–306.
9. O.H. Ibarra, A hierarchy theorem for polynomial-space recognition, *SIAM J. Comput.*, **3** 3 (1974) 184–187.
10. O.H. Ibarra, S.M.Kim, and S. Moran, Sequential machine characterizations of trellis and cellular automata and applications, *SIAM J. Comput.*, **14** 2 (1985) 426–447.
11. O.H. Ibarra and S.K. Sahni, Hierarchies of Turing machines with restricted tape alphabet size, *J. Comput. System Sci.*, **11** (1975) 56–67.
12. K. Iwama and C. Iwamoto, Parallel complexity hierarchies based on PRAMs and DLOGTIME-uniform circuits, *Proc. IEEE Conf. on Computational Complexity*, Philadelphia, 1996, 24–32.
13. C. Iwamoto, Complexity hierarchies on circuits under restricted uniformities, *Presentation at ICCI*, Kuwait, 2000.
14. C. Iwamoto, T. Hatsuyama, K. Morita, and K. Imai, Constructible functions in cellular automata and their applications to hierarchy results, *Theoret. Comput. Sci.*, **270** (2002) 797–809.
15. C. Iwamoto and K. Iwama, Time complexity hierarchies of extended TMs for parallel computation, *IEICE Trans. Inf. and Syst.*, **J80-D-I** 5 (1997) 421–427 (in Japanese).
16. C. Iwamoto and M. Margenstern, Time and space complexity classes of hyperbolic cellular automata, *IEICE Trans. on Inf. and Syst.*, **E87-D** 3 (2004) 265–273.
17. W.W. Kirchherr, A hierarchy theorem for PRAM-based complexity classes, *Proc. 8th Conf. on Foundations of Software Technology and Theoretical Computer Science (Lecture Notes in Computer Science)*, **338**, Pune, India, 1988, 240–249.
18. W.J. Paul, On time hierarchies, *J. Comput. System Sci.* **19** (1979) 197–202.
19. W.L. Ruzzo, On uniform circuit complexity, *J. Comput. System Sci.*, **22** (1981) 365–383.
20. M. Sipser, Borel sets and circuit complexity, *Proc. 15th Annual ACM Symp. on Theory of Computing*, Boston, Massachusetts, 1983, 61–69.
21. S. Žák, A Turing machine space hierarchy, *Kybernetika*, **26** 2 (1979) 100–121.
22. S. Žák, A Turing machine time hierarchy, *Theoret. Comput. Sci.*, **26** (1983) 327–333.

Several New Generalized Linear- and Optimum-Time Synchronization Algorithms for Two-Dimensional Rectangular Arrays

Hiroshi Umeo[*], Masaya Hisaoka, Masato Teraoka, and Masashi Maeda

Univ. of Osaka Electro-Communication,
Neyagawa-shi, Hatsu-cho 18-8, Osaka, 572-8530, Japan
umeo@umeolab.osakac.ac.jp

Abstract. We propose several new generalized synchronization algorithms for 2-D cellular arrays. Firstly, a generalized linear-time synchronization algorithm and its 14-state implementation are given. It is shown that there exists a 14-state 2-D CA that can synchronize any $m \times n$ rectangular array in $m+n+\max(r+s, m+n-r-s+2)-4$ steps with the general at an arbitrary initial position (r, s),where $1 \leq r \leq m$, $1 \leq s \leq n$. The generalized linear-time synchronization algorithm is interesting in that it includes an optimum-step synchronization algorithm as a special case where the general is located at one corner. In addition, we propose a noveloptimum-time generalized synchronization scheme that can synchronize any $m \times n$ array in $m+n+\max(m, n)-\min(r, m-r+1)-\min(s, n-s+1)-1$ optimum steps.

1 Introduction

We study a synchronization problem which gives a finite-state protocol for synchronizing a large scale of cellular automata. The synchronization in cellular automata has been known as firing squad synchronization problem since its development, in which it was originally proposed by J. Myhill to synchronize all parts of self-reproducing cellular automata [7]. The firing squad synchronization problem has been studied extensively for more than 40 years [1-16]. The present authors are involved in research on firing squad synchronization algorithms on two-dimensional (2-D) cellular arrays. Several synchronization algorithms on 2-D arrays have been proposed, including Grasselli [3], Kobayashi [4], Shinahr [10] and Szwerinski [12]. To date, the smallest number of cell states for which an optimum-time synchronization algorithm has been developed is 28 for rectangular array, achieved by Shinahr [10].

In this paper, several new generalized synchronization algorithms and their efficient implementations for 2-D cellular arrays will be given. In section 3, we propose a linear-time 14-state generalized synchronization algorithm that can synchronize any $m \times n$ rectangular array in $m+n+\max(r+s, m+n-r-$

[*] Corresponding author

$s+2)-4$ steps with the general at an arbitrary initial position (r,s). We show that our linear-time 14-state solution yields an optimum-time synchronization algorithm in the case where the general is located at one corner. We progressively reduce the number of internal states of each cellular automaton on rectangular arrays, achieving fourteen states. In section 4, we propose a new generalized optimum-time synchronization scheme that can synchronize any $m \times n$ array in $m+n+\max(m,n)-\min(r,m-r+1)-\min(s,n-s+1)-1$ optimum steps. The design scheme is based on freezing-thawing technique developed in Umeo [15]

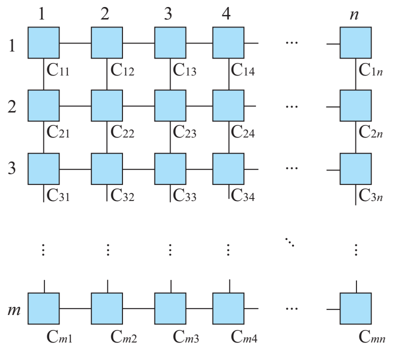


Fig. 1. A two-dimensional cellular automaton.

2 Firing Squad Synchronization Problem

Figure 1 shows a finite two-dimensional (2-D) cellular array consisting of $m \times n$ cells. Each cell is an identical (except the border cells) finite-state automaton. The array operates in lock-step mode in such a way that the next state of each cell (except border cells) is determined by both its own present state and the present states of its north, south, east and west neighbors. All cells (*soldiers*), except the north-west corner cell (*general*), are initially in the quiescent state at time $t=0$ with the property that the next state of a quiescent cell with quiescent neighbors is the quiescent state again. At time $t=0$, the north-west corner cell $C_{1,1}$ is in the *fire-when-ready* state, which is the initiation signal for synchronizing the array. The firing squad synchronization problem is to determine a description (state set and next-state function) for cells that ensures all cells enter the *fire* state at exactly the same time and for the first time. The set of states must be independent of m and n. We call the synchronization problem *normal*, when the initial position of the general is restricted to north-west corner of the array. In section 3 and 4 we consider a *generalized* firing squad synchronization problem, in which the general can be initially located at any position on the array. As for the normal synchronization problem, several algorithms have been proposed, including Beyer [2], Grasselli [3], Kobayashi [4], Shinahr [10], Szwerinski [12] and Umeo, Maeda and Fujiwara [13]. Umeo, Maeda and Fujiwara [13] presented a 6-state two-dimensional synchronization algorithm that fires any $m \times n$ arrays in $2(m+n)-4$ steps. The algorithm is slightly slower than the optimum ones, but the number of internal states is considerably smaller. Beyer [2] and Shinahr [10] presented an optimum-time synchronization scheme in order to synchronize any $m \times n$ arrays in $m+n+max(m,n)-3$ steps. To date, the smallest number of cell states for which an optimum-time synchronization algorithm has been developed is 28 for rectangular array, achieved by Shinahr [10]. On the other hand, it has been an only one algorithm that Szwerinski [12] proposed an optimum-time generalized 2-D firing algorithm with 25,600 internal states.

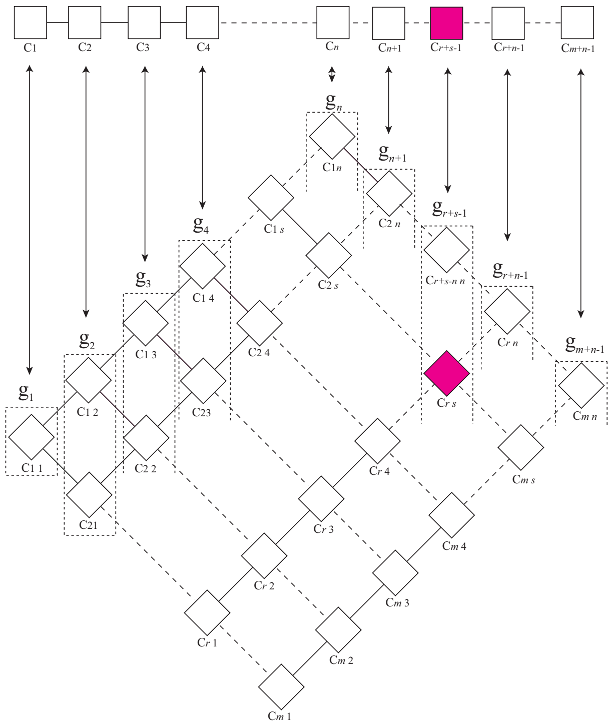


Fig. 2. Generalized correspondence between 1-D and 2-D cellular arrays.

3 A Generalized Linear-Time Synchronization Algorithm

Now we consider a generalized firing squad synchronization problem, in which the general can be initially located at any position on the array. Before presenting

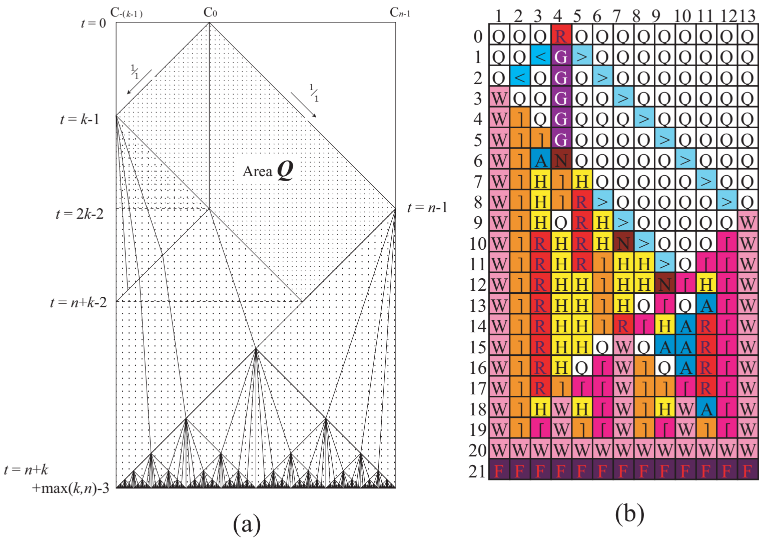


Fig. 3. Time-space diagram for generalized optimum-step firing squad synchronization algorithm (Fig. 3(a)) and snapshots for a 12-state implementation of generalized firing squad synchronization algorithm with the property $\mathcal{Q}$ on 13 cells with a general on C_4 (Fig. 3(b)).

the algorithm, we propose a simpler mapping scheme, which is different from the previous one, for embedding one-dimensional generalized synchronization algorithms onto two-dimensional arrays. Consider a 2-D array of size $m \times n$. We divide mn cells into $m+n-1$ groups g_k, $1 \leq k \leq m+n-1$, defined as follows.

$$g_k = \{C_{i,j} | (i-1) + (j-1) = k-1\}, \text{ i.e.,}$$

$g_1 = \{C_{1,1}\}$, $g_2 = \{C_{1,2}, C_{2,1}\}$, $g_3 = \{C_{1,3}, C_{2,2}, C_{3,1}\}$, . . . , $g_{m+n-1} = \{C_{m,n}\}$. Figure 2 shows the division into $m+n-1$ groups.

Property $\mathcal{Q}$: We say that a generalized firing algorithm has a *property* $\mathcal{Q}$, where any cell, except the general C_k, keeps a quiescent state *in the zone* Q of the time-space diagram shown in Fig. 3(a).

The one-dimensional generalized firing squad synchronization algorithm with the *property* $\mathcal{Q}$ can be easily embedded onto two-dimensional arrays with a small overhead. Fig. 3(b) shows snapshots of our 12-state optimum-time generalized firing squad synchronization algorithm with the *property* $\mathcal{Q}$.

[Theorem 1] There exists a 12-state one-dimensional CA with the *property* $\mathcal{Q}$ which can synchronize n cells in exactly optimum $n-2+\max(k, n-k+1)$ steps, where the general is located on C_k.

For any 2-D array M of size $m \times n$ with the general at $C_{r,s}$, where $1 \leq r \leq m$, $1 \leq s \leq n$, there exists a corresponding 1-D cellular array N of length $m+n-1$ with the general at C_{r+s-1} such that the configuration of N can be mapped on

M, and M fires if and only if N fires. Let S^t_i, $S^t_{i,j}$ and $S^t_{g_i}$ denote the state of C_i, $C_{i,j}$ at step t and the set of states of the cells in g_i at step t, respectively. Then, we can establish the following lemma.

[Lemma 2] The following two statements hold:

1. For any integer i and t such that $1 \leq i \leq m+n-r-s+1$, $r+s+i-3 \leq t \leq T(m+n-1, r+s-1)$, $\parallel S^t_{g_i} \parallel = 1$ and $S^t_{g_i} = S^t_i$. That is, all cells in g_i at step t are in the same state and it is equal to S^t_i, where the state in $S^t_{g_i}$ is simply denoted by $S^t_{g_i}$.
2. For any integer i and t such that $m+n-r-s+2 \leq i \leq m+n-1$, $2m+2n-r-s-i-1 \leq t \leq T(m+n-1, r+s-1)$, $\parallel S^t_{g_i} \parallel = 1$ and $S^t_{g_i} = S^t_i$.

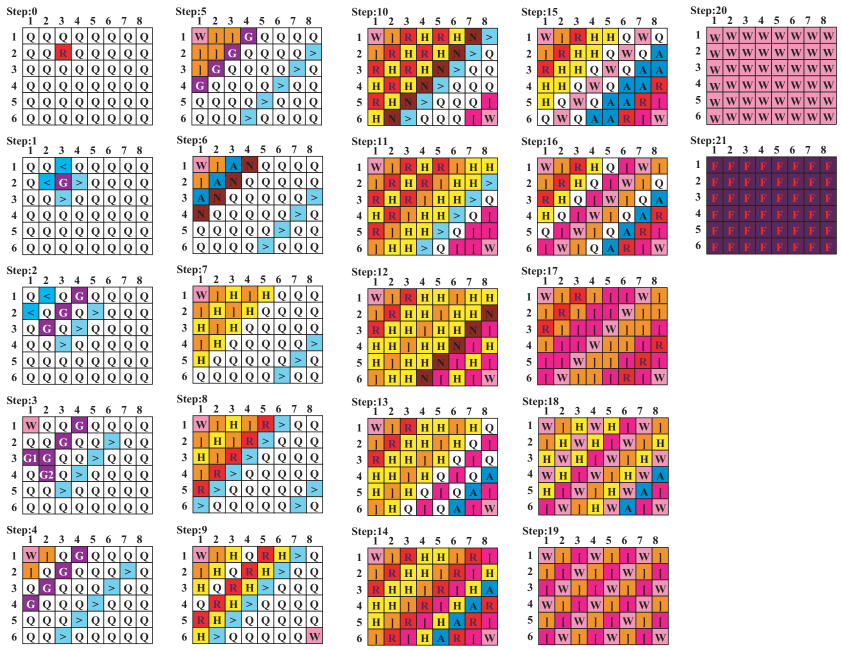


Fig. 4. Snapshots of our 14-state linear-time generalized firing squad synchronization algorithm on rectangular arrays.

Based on the 12-state generalized 1-D algorithm given above, we obtain the following 2-D generalized synchronization algorithm that synchronizes any 2-D array of size $m \times n$ in $m+n-1-2+\max(r+s-1, m+n-r-s+1) = m+n+\max(r+s, m+n-r-s+2)-4$ steps. Two additional states are required in our construction (details omitted). Szwerinski [12] also proposed an optimum-time generalized 2-D firing algorithm with 25,600 internal states that fires any

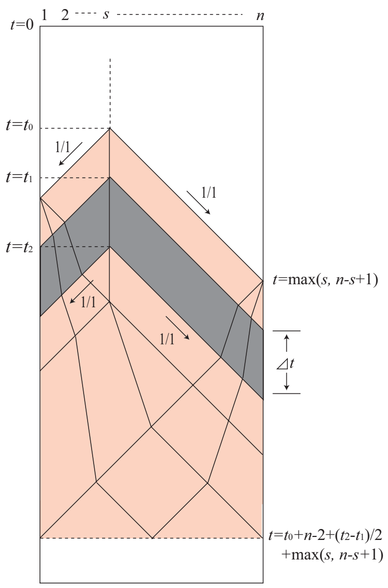


Fig. 5. Time-space diagram for delayed optimum-time generalized synchronization algorithm.

$m \times n$ array in $m+n+\max(m,n)-\min(r,m-r+1)-\min(s,n-s+1)-1$ steps, where (r,s) is the general's initial position. Our 2-D generalized synchronization algorithm is $\max(r+s,m+n-r-s+2)-\max(m,n)+\min(r,m-r+1)+\min(s,n-s+1)-3$ steps larger than the optimum algorithm proposed by Szwerinski [12]. However, the number of internal states required to yield the firing condition is the smallest known at present. Snapshots of our 14-state generalized synchronization algorithm running on a rectangular array of size 6×8 with the general at $C_{3,4}$ are shown in Fig. 4.

[**Theorem 3**] There exists a 14-state 2-D CA that can synchronize any $m \times n$ rectangular array in $m+n+\max(r+s,m+n-r-s+2)-4$ steps with the general at an arbitrary initial position (r,s).

Our linear-time synchronization algorithm is interesting in that it includes an optimum-step synchronization algorithm as a special case where the general is located at the north-east corner. By letting $r=1$, $s=n$, we get $m+n+\max(r+s,m+n-r-s+2)-4 = m+n+\max(n+1,m+1)-4 = m+n+\max(m,n)-3$. Thus the algorithm is a time-optimum one. Then, we have:

[**Theorem 4**] There exists a 14-state 2-D CA that can synchronize any $m \times n$ rectangular array in $m+n+\max(m,n)-3$ steps.

4 A Generalized Time-Optimum Firing Squad Synchronization Agorithm

In this section we propose a novel optimum-time generalized synchronization scheme for two-dimensional rectangular arrays of size $m \times n$ with the general being located at any position (r, s) on the array. The following observation is a useful technique for delaying the generalized synchronization on one-dimensional arrays.

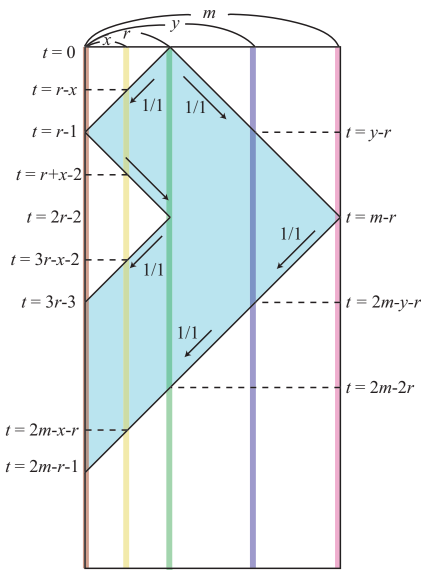


Fig. 6. Time-space diagram for delaying row synchronization.

[Observation 5] Let A be any one-dimensional cellular automaton that runs a generalized $T(s,n)$-step synchronization algorithm on n cells with an initial general on $\mathrm{C}_s(1 \leq s \leq n)$ and t_0, t_1, t_2, ℓ be any integer satisfying the following conditions such that $\Delta t = t_2 - t_1 = 2\ell$, $\ell \geq 1$, $t_2 > t_1 \geq t_0 \geq 0$ and $t_1 + t_2 - 2t_0 \leq 2T(s,n) - 2\max(s, n-s+1) + 2$. We also assume that three special signals are given to cell C_s at step $t = t_0$, t_1, and t_2. These signals play an important role of initializing the generalized synchronization process, starting the delayed operation, and stopping the delayed operation, respectively. Then, we can construct a cellular automaton B that synchronizes the array at time $t = t_0 + \ell + T(s,n)$. In the case where $T(s,n)$ is an optimum time complexity such that $T(s,n) = n - 2 + \max(s, n-s+1)$, the constructed B fires at time $t = t_0 + \ell + n - 2 + \max(s, n-s+1)$. Thus the synchronization operation is delayed for ℓ steps. Figure 5 is a time-space diagram for the delayed optimum-time generalized synchronization algorithm operating on n cells. In the darker area of the diagram, each cell simulates the operation of A at speed $1/2$ by repeating a *simulate-one-step* of A then *keep-the-state* operations alternatively at each step.

Without loss of generality, we assume that $m \le n$, $1 \le r < \lceil m/2 \rceil$ and $1 \le s < \lceil n/2 \rceil$. We regard an array of size $m \times n$ as consisting of independent m rows of length n. An optimum-time generalized synchronization algorithm with a general at $\mathrm{C}_{i,s}(1 \le i \le m)$ is used for the synchronization of the i-th row. We call the operations *row-synchronization.* To fire all rows simultaneously, an efficient timing control scheme shown in Fig. 6 is developed. Figure 6 is a time-space diagram for giving special signals to each cell on the s-column. These special signals act as a timing $t = t_0$, t_1 and t_2 stated in [Observation 5]. For example, the row-synchronization on the y-th $(r \le y \le m)$ row is started at time $t = t_0 = t_1 = y - r$ and the process is delayed from time $t = t_1 = y - r$ to $t = t_2 = 2m - y - r$, shown in the darker area in Fig. 6. Thus $\ell = m - y$. Based on [Observation 5] the y-th row is fired at time $t = m + 2n - r - s - 1$. Figure 12 is a time-space diagram for the row-synchronization on the x-th and y-th row, where $1 \le x < r$ and $r \le y \le m$. The readers can see how all rows are synchronized at time $t = m + 2n - r - s - 1$.

Thus we have:

[Lemma 6] In the row-synchronization process, all of the rows can be fired simultaneously at time $t = m + 2n - r - s - 1$ in the case $m \le n$, $1 \le r < \lceil m/2 \rceil$ and $1 \le s < \lceil n/2 \rceil$. In the column-synchronization process, all of the cells take a firing state prior to the row-synchronization, but the column-synchronization fails to synchronize.

Symmetrically we get the following lemma in the case where the array is longer than is wide.

[Lemma 7] In the column-synchronization process, all of the columns can be fired simultaneously at time $t = n + 2m - r - s - 1$ in the case $m \ge n$, $1 \le r < \lceil m/2 \rceil$ and $1 \le s < \lceil n/2 \rceil$. In the row-synchronization process, all of the cells take a firing state prior to the column-synchronization, but the row-synchronization fails to synchronize.

To synchronize the array in optimum-time, the array performs both row- and column-synchronization operations. Each cell takes a firing state at two different times. The first one is false and it should be ignored. The second one is a right firing state. By combining the [Lemmas 6, 7], we can establish the following theorem.

[Theorem 8] The scheme given above can synchronize any $m \times n$ array in $m + n + \max(m, n) - \min(r, m - r + 1) - \min(s, n - s + 1) - 1$ optimum steps, where (r, s) is the general's initial position.

5 Conclusions

We have proposed several new generalized synchronization algorithms and their state-efficient implementations for 2-D cellular arrays. The first linear-time algo-

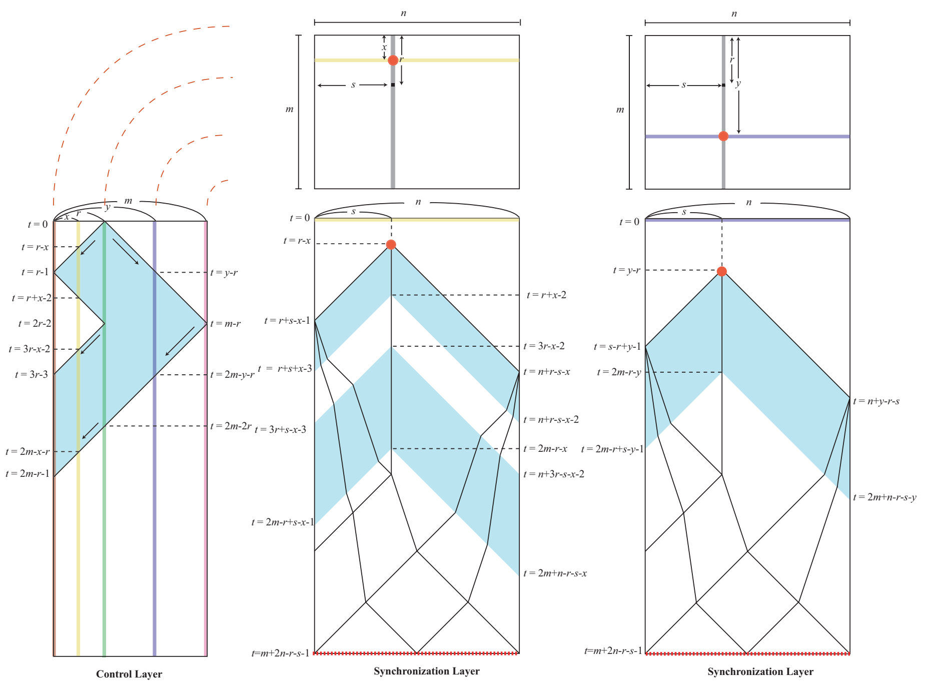


Fig. 7. Time-space diagram for the row-synchronization on the x-th and y-th row of a rectangular array of size $m \times n$, where $m \leq n$, $1 \leq r < \lceil m/2 \rceil$, $1 \leq s < \lceil n/2 \rceil$, $1 \leq x < r$ and $r \leq y \leq m$.

rithm is based on an efficient mapping scheme, achieving fourteen state operating in $m+n+\max(r+s, m+n-r-s+2)-4$ steps for any 2-D rectangular array of size $m \times n$ with the general at an arbitrary initial position (r, s), where $1 \leq r \leq m$, $1 \leq s \leq n$. The generalized linear-time synchronization algorithm proposed is interesting in that it includes an optimum-step synchronization algorithm as a special case where the general is located at one corner. Lastly, we have proposed a new optimum-time generalized synchronization scheme that can synchronize any $m \times n$ array in $m+n+\max(m, n)-\min(r, m-r+1)-\min(s, n-s+1)-1$ optimum steps. It is an interesting question that how many states are needed for its implementation.

References

1. R. Balzer: An 8-state minimal time solution to the firing squad synchronization problem. *Information and Control*, vol. 10(1967), pp. 22-42.
2. W. T. Beyer: Recognition of topological invariants by iterative arrays. Ph.D. Thesis, MIT, (1969), pp. 144.

3. A. Grasselli: Synchronization of cellular arrays: The firing squad problem in two dimensions. *Information and Control*, vol. 28(1975), pp. 113-124.
4. K. Kobayashi: The firing squad synchronization problem for two-dimensional arrays. *Information and Control*, vol. 34(1977), pp. 177-197.
5. J. Mazoyer: A six-state minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, vol. 50(1987), pp. 183-238.
6. M. Minsky: *Computation: Finite and infinite machines.* Prentice Hall, (1967), pp. 28-29.
7. E. F. Moore: The firing squad synchronization problem. in *Sequential Machines, Selected Papers* (E. F. Moore ed.), Addison-Wesley, Reading MA., (1964), pp. 213-214.
8. F. R. Moore and G. G. Langdon: A generalized firing squad problem. *Information and Control*, vol. 12(1968), pp. 212-220.
9. H. B. Nguyen and V. C. Hamacher: Pattern synchronization in two-dimensional cellular space. *Information and Control*, vol. 26(1974), pp. 12-23.
10. I. Shinahr: Two- and three-dimensional firing squad synchronization problems. *Information and Control*, vol. 24(1974), pp. 163-180.
11. A. Settle and J. Simon: Smaller solutions for the firing squad. *Theoretical Computer Science*, vol. 276(2002), pp.83-109.
12. H. Szwerinski: Time-optimum solution of the firing-squad-synchronization-problem for n-dimensional rectangles with the general at an arbitrary position. *Theoretical Computer Science*, vol. 19(1982), pp. 305-320.
13. H. Umeo, M. Maeda and N. Fujiwara: An efficient mapping scheme for embedding any one-dimensional firing squad synchronization algorithm onto two-dimensional arrays. *Proc. of the 5th International Conference on Cellular Automata for Research and Industry*, LNCS 2493, Springer-Verlag, pp.69-81(2002).
14. H. Umeo, M. Hisaoka, K. Michisaka, K. Nishioka and M. Maeda: Some generalized synchronization algorithms and their implementations for a large scale cellular automata. *Proc. of the Third International Conference on Unconventional Models of Computation, UMC 2002*, LNCS 2509, Springer-Verlag, pp.276-286(2002).
15. H. Umeo: A simple design of time-efficient firing squad synchronization algorithms with fault-tolerance. *IEICE Trans. on Information and Systems*, vol.E-87-D, No.3(2004), pp.733-739.
16. A. Waksman: An optimum solution to the firing squad synchronization problem. *Information and Control*, vol. 9(1966), pp. 66-78.

Register Complexity of LOOP-, WHILE-, and GOTO-Programs

Markus Holzer[1] and Martin Kutrib[2]

[1] Institut für Informatik, Technische Universität München,
Boltzmannstraße 3, D-85748 Garching bei München, Germany
holzer@informatik.tu-muenchen.de
[2] Institut für Informatik, Universität Gießen,
Arndtstraße 2, D-35392 Gießen, Germany
kutrib@informatik.uni-giessen.de

Abstract. We study the register complexity of LOOP-, WHILE-, and GOTO-programs, that is the number of registers (or variables) needed to compute certain unary (partial) functions from the non-negative integers to the non-negative integers. It turns out that the hierarchy of LOOP-computable (WHILE-, and GOTO-computable, respectively) functions $f : \mathbb{N}_0 \to \mathbb{N}_0$ (partial functions $f : \mathbb{N}_0 \hookrightarrow \mathbb{N}_0$, respectively) that is induced by the number of registers collapses to a fixed level. In all three cases the first levels are separated. Our results show that there exist universal WHILE- and GOTO-programs with a constant number of registers.

1 Introduction

Computability theory dates back to the beginning of the last century and is at the heart of theoretical computer science. From the early days the field has grown rapidly and has initiated many new branches in computer science theory, like abstract complexity theory, domain theory and λ-calculus, etc. Several characterizations of computable functions—see, e.g., [1,6,9,10,11,12]—have been proposed and all of them were shown to be equivalent. Here we are interested in the programming language approach to computability by WHILE- and GOTO-programs as proposed in, e.g., [2].

It is known that WHILE- and GOTO-programs precisely characterize the μ-recursive or partial recursive functions and, hence, are universal in the computational sense. Primitive recursive functions are also representable by a programming language approach, namely by so called LOOP-programs [7]. Compared to the WHILE-statement, where the body is evaluated as long as the condition on the specified register or variable is true, a LOOP-statement evaluates its body m times, if the specified register holds the value m before entering the loop. Therefore, LOOP-programs can only compute total functions, and thus do not encompass everything that we think of as computable. Nevertheless, they form an important class and many of these functions are normally studied in number theory.

A natural measure of complexity for LOOP-programs is the nesting depth of the iteration statements. It induces an infinite hierarchy of function classes, which coincides with the Axt and Grzegorczyk classes [4] for large enough nesting depths. An immediate consequence of the nesting depth non-collapse result is that there are no universal `LOOP`-programs for the family of primitive recursive functions.

In this paper we study another natural measure of complexity for all three program types, namely the number of registers needed to compute a function. It turns out that the hierarchies of function classes induced by the number of registers collapse to fixed levels for `LOOP`-, `WHILE`- and `GOTO`-programs. This nicely contrasts the situation for the nesting depth hierarchy of `LOOP`-programs. In case of `GOTO`- and `WHILE`-programs we can show that three register are sufficient to simulate every partial recursive function. For `LOOP`-programs four registers are sufficient to simulate any `LOOP`-program with an arbitrary number of registers. Whether these results are optimal have been left open. Nevertheless, the first levels of the register hierarchies are separated.

The paper is organized as follows: The next section contains preliminaries. Section 3 is devoted to register complexity of `LOOP`-programs and in Section 4 we deal with `WHILE`- and `GOTO`-programs, showing that in all three cases the hierarchies on the number of registers collapse to fixed levels. Finally, we summarize our results and discuss some open questions in Section 5.

2 Definitions

We assume the reader to be familiar with the basic notions of recursion and computability theory as contained in [5].

Let $\mathbb{N}_0$ denote the non-negative integers (natural numbers). A `WHILE`-program is a finite sequence of commands for changing natural numbers stored in registers. There is no limit to the size of an integer which may be stored in a register. In general, there is also no limit to the number of registers to which a program may refer, although any given program will refer to only a fixed number of registers. The registers are identified by names. The syntax of `WHILE`-programs is as follows:

⟨program⟩	::=	⟨command⟩	
	$\mid$	⟨command⟩ `;` ⟨program⟩	
⟨command⟩	::=	⟨expression⟩	
	$\mid$	`WHILE` ⟨condition⟩ `DO` ⟨program⟩ `OD`	
⟨expression⟩	::=	⟨variable⟩ := 0	
	$\mid$	⟨variable⟩ := ⟨variable⟩ + 1	(both variable names
	$\mid$	⟨variable⟩ := ⟨variable⟩ − 1	must be the same)
⟨condition⟩	::=	⟨variable⟩ = 0	
	$\mid$	⟨variable⟩ $\neq$ 0	
⟨variable⟩	::=	$x \mid y \mid \ldots$	

The operations of assignment, successor, and predecessor change the contents of registers or variables. In case of initialization (successor, predecessor, respectively) the content of the register or variable is set to zero (is incremented by one, decremented by one, respectively). Note that the predecessor of a register that contains zero remains zero—this is the modified subtraction. The rest of the semantics, in particular the WHILE-statement is natural and so not presented here. Observe, that assignments of the form, e.g., $x := y + 1$, are not allowed and because of the conditions $= 0$ and $\neq 0$, a register content can be checked to be zero or non-zero, respectively. In the forthcoming, the content (value) of a register x will be referred to as x.

A WHILE-program P *computes* or defines a partial function $\varphi_P : \mathbb{N}_0 \hookrightarrow \mathbb{N}_0$ as follows: Initially the input n is stored in the input register and all other registers are initialized with zero. When P terminates, then the value of φ_P on input n is given by the value of the output register. If not otherwise stated, we set x (or c_1) to be the input and output register. If the program does not terminate, then $\varphi_P(n)$ is undefined. For convenience we simply write φ instead of φ_P. A partial function $f : \mathbb{N}_0 \hookrightarrow \mathbb{N}_0$ has WHILE-*program register complexity* k, for some $k \geq 1$, if and only if there is a WHILE-program P using k registers such that $f(n) = \varphi_P(n)$, for all n in $\mathbb{N}_0$. The family of partial functions that are WHILE-computable with register complexity k is denoted by $\mathtt{WHILE}_k$. Moreover, $\mathtt{WHILE} := \bigcup_{k\in\mathbb{N}} \mathtt{WHILE}_k$ is the class of all partial functions.

An example of a WHILE-program, called `non-terminate`, that defines the overall undefined function is:

```
$x_1 := 0$;  $x_1 := x_1 + 1$;
WHILE $x_1 \neq 0$ DO $x_1 := x_1 + 1$ OD
```

The syntax of LOOP-programs is similarly defined as for WHILE-programs, except that the command LOOP ⟨variable⟩ DO ⟨program⟩ OD is used instead of WHILE ⟨condition⟩ DO ⟨program⟩ OD. The semantics of the LOOP-statement is as follows: The body of the LOOP-statement is evaluated m times, if the specified register holds value m before entering the loop. Observe, that LOOP-programs always define total functions. The family of functions that are LOOP-computable with register complexity k is denoted by $\mathtt{LOOP}_k$, and $\mathtt{LOOP} := \bigcup_{k\in\mathbb{N}} \mathtt{LOOP}_k$.

In the remainder of this section we discuss the syntax of GOTO-programs. Sometimes it is convenient for programming to use a lower-level flow chart syntax, in which a program is a sequence $M_1 : P_1;\ M_2 : P_2; \ldots ; M_n : P_n$ of labeled (and unlabeled) commands, executed sequentially except for explicitly redirected control transfer. Commands and labels are of the form:

⟨command⟩	::=	⟨pcommand⟩
	|	⟨label⟩ : ⟨pcommand⟩
⟨pcommand⟩	::=	⟨expression⟩
	|	GOTO ⟨label⟩
	|	IF ⟨condition⟩ GOTO ⟨label⟩
⟨label⟩	::=	$M_1 \mid M_2 \mid \ldots$

The GOTO-equivalent of the `non-terminate` program is M_1 : GOTO M_1. Though the name of the input register may not appear in the source code, it still counts as a used register. Therefore, the GOTO-program register complexity of the above shown GOTO-program equals 1. Finally, we denote the family of partial functions with GOTO-program register complexity k by $\texttt{GOTO}_k$ and set $\texttt{GOTO} := \bigcup_{k\in\mathbb{N}} \texttt{GOTO}_k$.

By definition of $\texttt{LOOP}_k$, $\texttt{WHILE}_k$, and $\texttt{GOTO}_k$ we have the following trivial chain of inclusions: $\texttt{LOOP}_1 \subseteq \texttt{LOOP}_2 \subseteq \cdots \subseteq \texttt{LOOP}_i \subseteq \texttt{LOOP}_{i+1} \subseteq \cdots \subseteq \texttt{LOOP}$, which also holds for WHILE-computable and GOTO-computable functions.

3 LOOP-Programs

As already mentioned, LOOP-programs in general are equivalent in computational power to primitive recursive functions. In the following we show that the register complexity of LOOP-programs can always be reduced to four. The main idea is similar to that presented in [8], where it is shown that a 2-counter automaton is universal.

Theorem 1. *Let $f : \mathbb{N}_0 \to \mathbb{N}_0$ be a function that is computable by a* LOOP*-program P with an arbitrary number of registers. Then there is a* LOOP*-program P' with register complexity four, such that $\varphi_{P'}(n) = f(n)$, for all $n \in \mathbb{N}_0$.*

Proof. The idea for the simulation of a LOOP-program P with an arbitrary number of registers by a LOOP-program with a constant number of registers is to encode the values of all used registers into one register and to update this encoding appropriately according to the instructions from the original program P. Let k be the register complexity of the LOOP-program P. The values of the register $x_1, x_2, \ldots, x_k$ of P will be encoded by the number $p_1^{x_1} p_2^{x_2} \ldots p_k^{x_k}$, where the p_i's, for $1 \le i \le k$, are mutually distinct prime numbers. Before we construct a LOOP-program with four registers we need some code-fragments: The sub-program power-p reads as follows.

```
power-p($x, y$):                    {computes $p^x$ for some constant $p$}
    $y := 0$;  $y := y + 1$;
    LOOP $x$ DO multiply-p($y$) OD;
    $x := 0$;  LOOP $y$ DO $x := x + 1$ OD;
```

where the program code for multiply-p is:

```
multiply-p($x$):                    {computes $p \cdot x$ for some constant $p$}
    LOOP $x$ DO $x := x + p - 1$ OD;
```

```
divide-p($x, y, z$):                {computes $\frac{x}{p}$ if $p$ divides $x$}
    $y := 0$;  LOOP $x$ DO $y := y + 1$ OD;
    div-p($y, z$);  multiply-p($y$);
    $z := 0$;  LOOP $x$ DO $z := z + 1$ OD;
    equals($y, z$);
    LOOP $y$ DO div-p($x, y$) OD;
```

computes $\frac{x}{p}$ on input x, if p divides x. In this case, register y has also the content $\frac{x}{p}$ which is always greater than zero. If otherwise p does not divide x, the input is not changed and register y has content zero. The program codes for div-p and equals read as:

```
div-p($x, y$):                    {computes $\lfloor \frac{x}{p} \rfloor$ for some constant $p$}
    $y := 0$;    $x := x + 1$;
    LOOP $x$ DO
      $x := x - p$;               {implement it by iterated subtraction}
      LOOP $x$ DO $y := y + 1$ OD;
      $x := x - 1$;
      LOOP $x$ DO $y := y - 1$ OD;
      $x := x + 1$
    OD;
    $x := 0$; LOOP $y$ DO $x := x + 1$ OD;
```

and

```
equals($x, y$):                   {compares $x$ and $y$ under
    LOOP $x$ DO $y := y - 1$ OD;  the assumption that $x \leq y$}
    $x := 0$; $x := x + 1$;
    LOOP $y$ DO $x := 0$ OD;
```

computes 1 if $x = y$, and 0 if $x < y$. Finally, we need

```
extract-p($x, y, z, r$):          {computes largest $n$ such
    $r := 0$;                     that $p^n$ divides $x$ exactly}
    LOOP $x$ DO
      divide-p($x, y, z$);
      $r := r + 1$;    $z := 0$;
      equals($y, z$);
      LOOP $y$ DO $r := r - 1$ OD
    OD;
```

that computes on input x the largest n such that p^n divides x exactly. The result is stored in register r.

Now we are ready to construct a LOOP-program with registers x, y, z, r from the given program P as follows: First we encode the original input to its appropriate form $p_1^{x_1}$, i.e., we start our program code with power-p(x, y); where the result is stored in register x. Then each command of P is rewritten as follows—we distinguish four cases:

(1) Command $x_i := 0$; is replaced by LOOP x_i DO $x_i := x_i - 1$ OD; in the *original* program code—thus, this case is reduced to cases below.
(2) Command $x_i := x_i + 1$; is simulated by multiply-p(x); where prime p_i is used for p.
(3) Command $x_i := x_i - 1$; is simulated by divide-p(x, y, z); where prime p_i is used for p.
(4) For the command LOOP x_i DO Q OD; one has to extract the value of x_i from the encoding. The program line LOOP x_i DO Q OD; is simulated by

```
    extract-p($x, y, z, r$);
    LOOP $r$ DO multiply-p($x$) OD;
    LOOP $r$ DO $Q'$ OD;
```

where prime p_i is used for p and Q' is the program that is built by induction from Q.

Finally, we have to reconstruct the value of the output register. To this end, we run `extract-p`(x, y, z, r)`;` $x := 0$`;` `LOOP` r `DO` $x := x + 1$ `OD;` where prime p_1 is used for p. This completes the description of the program and shows that any `LOOP`-program can be simulated by another `LOOP`-program with four registers only. □

Corollary 2. $\mathtt{LOOP} = \mathtt{LOOP}_4$.

The following lemma classifies the functions that belong to $\mathtt{LOOP}_1$. Recall that in terms of orders of magnitude, function $f : \mathbb{N}_0 \to \mathbb{N}_0$ is a lower bound of the set $\Omega(f) = \{g : \mathbb{N}_0 \to \mathbb{N}_0 \mid \exists\, n_0, c \in \mathbb{N} : \forall\, n \geq n_0 : c \cdot g(n) \geq f(n)\}$.

Lemma 3. *Let P be a `LOOP`-program with one register. Then there exist constants $a \geq 1$, $b \in \mathbb{Z}$, and $c > 1$ such that either (i) $\varphi_P(n) \leq c$, (ii) $\varphi_P(n) = an+b$, or (iii) $\varphi_P(n) \in \Omega(c^n)$.*

Proof. The statement is shown by induction on the structure of `LOOP`-programs.

(1) Let P be one of the expressions $x := x+1$, $x := x-1$, or $x := 0$, respectively. Then it follows immediately $\varphi_P(n) = an + b$, for $a = b = 1$, $\varphi_P(n) = an + b$, for $a = 1$, $b = -1$, or $\varphi_P(n) \leq c$, for $c = 0$, respectively.

(2) Let P be the command sequence $P_1; P_2$, where the assertion has been shown for φ_{P_1} and φ_{P_2}. In general, $\varphi_P(n)$ is the composition of φ_{P_1} with φ_{P_2}. We have to distinguish three sub-cases.

   (a) If $\varphi_{P_1}(n) \leq c$, then φ_P is approximated by applying φ_{P_2} to c, i.e., $\varphi_P(n) \leq \varphi_{P_2}(c)$. Therefore, $\varphi_P(n) \leq c'$, for some constant c'.

   (b) For the case $\varphi_{P_1}(n)$ is of form $a_1 n + b_1$, the function φ_{P_2} determines the order of φ_P. In particular, (i) if $\varphi_{P_2}(n) \leq c$, then $\varphi_P(n) \leq c$, (ii) if $\varphi_{P_2}(n) = a_2 n + b_2$, then $\varphi_P(n) = a_2(a_1 n + b_1) + b_2 = a_2 a_1 n + a_2 b_1 + b_2$. Setting $a = a_2 a_1$ and $b = a_2 b_1 + b_2$ we obtain $\varphi_P(n) = an + b$. (iii) If $\varphi_{P_2}(n)$ is of order $\Omega(c^n)$, then $\varphi_P(n)$ is of order $\Omega(c^{a_1 n + b_1})$. Since $a_1 \geq 1$, this is of order $\Omega(c^n)$ also.

   (c) Now let $\varphi_{P_1}(n)$ be of order $\Omega(c_1^n)$. If $\varphi_{P_2}(n) \leq c$, then $\varphi_P(n) \leq c$. If $\varphi_{P_2}(n) = an + b$, then we have $\varphi_P(n) = a\varphi_{P_1}(n) + b$. Since $a \geq 1$ this is of order $\Omega(c_1^n)$ also. Finally, let $\varphi_{P_2}(n)$ be of order $\Omega(c_2^n)$. Clearly, $\varphi_P(n) = \varphi_{P_2}(\varphi_{P_1}(n))$ is of order $\Omega(c_1^n)$.

(3) Let P be the command `LOOP` x `DO` P_1 `OD`, where the assertion has been shown for φ_{P_1}. Let the value of x be n. Then in general, we obtain $\varphi_P(n)$ to be the n-fold composition of $\varphi_{P_1}(n)$, i.e., $\varphi_P(n) = \varphi_{P_1}(\varphi_{P_1}(\ldots \varphi_{P_1}(n)))$. Again, we distinguish three sub-cases as follows.

   (a) If $\varphi_{P_1}(n) \leq c$, then $\varphi_P \leq c'$, for some $c' \geq 1$, follows immediately.

   (b) For the case $\varphi_{P_1}(n) = an+b$ we derive $\varphi_P(n) = a^n n + a^{n-1} b + \cdots + ab + b$. We distinguish three cases: (i) If $a = 1$ and $b \geq 0$, this equals $n + bn = (b+1)n$, and for $a' = b+1$ and $b' = 0$ we obtain $\varphi_P(n) = a'n + b'$. (ii)

If $a = 1$ and $b < 0$, the result is $n - bn$. Since b is an integer, $\varphi_P(n) = 0$ follows. (iii) Finally, if $a > 1$, the result is clearly at least of order $\Omega(a^n)$, for all $b \in \mathbb{Z}$.

(c) The last case concerns $\varphi_{P_1}(n)$ of order $\Omega(c^n)$. The n-fold composition of φ_{P_1} does not decrease the order of magnitude in this case. Thus, $\varphi_P(n)$ is of order $\Omega(c^n)$. □

As an immediate consequence of the previous lemma, we can separate the first levels of the LOOP-program register hierarchy.

Theorem 4. $\mathtt{LOOP}_1 \subset \mathtt{LOOP}_2$

Proof. As witness function consider $f(n) = n^2$. By Lemma 3 function f cannot be computed by any LOOP-program with one register. It is straightforward to construct a LOOP-program P with two registers, such that $\varphi_P = f$. □

So we have $\mathtt{LOOP}_1 \subset \mathtt{LOOP}_2 \subseteq \mathtt{LOOP}_3 \subseteq \mathtt{LOOP}_4 = \cdots = \mathtt{LOOP}_i = \cdots = \mathtt{LOOP}$.

4 WHILE- and GOTO-Programs

This section is devoted to the study of the register hierarchy of WHILE- and GOTO-programs. Both program types are known to characterize all partial recursive functions. Before we continue with our investigations, we have to define counter machines, since they will be used for our WHILE- and GOTO-program simulations.

It is well known that there exist universal machines with two counters [8]. The input to those machines is supplied on an extra input tape. The counter machines as defined below get their input in one of the counters. For this mode it is an open question and unlikely that there exists a universal two-counter machine [3]. The problem is to encode the input x to 2^x and to decode the output 2^x to x. If this would be possible with only two counters, then there would be a universal two counter machine. On the other hand, this task is trivial for three counters.

A *k-counter machine* (without extra input tape) is a finite state device equipped with $k \in \mathbb{N}$ counters that may have values from the natural numbers. The machine can perform zero-tests on its counters. A counter can be incremented by 1, decremented by 1, or left unchanged. So, the instructions of the machine are given by a partial mapping $\delta : S \times \{0,1\}^k \to S \times \{+, n, -\}^k$, where S denotes the finite set of states, the results of the zero-tests are 0 or 1, and $+$, n, $-$ denote the operations on the counters. Depending on the current state and the contents of the counters, the machine changes to a new state and possibly manipulates the counters. Initially, the input is stored in the first counter, the machine is in the distinguished initial state, and the other counters are zero. The computation stops when the mapping δ is not defined for the current situation. In this case we say that the computation result is defined and is stored in the first counter. So, a k-counter machine A computes a (partial) function $\gamma_A : \mathbb{N}_0 \hookrightarrow \mathbb{N}_0$.

4.1 WHILE-Programs

Theorem 5. *Let $f : \mathbb{N}_0 \hookrightarrow \mathbb{N}_0$ be a function that is computable by a* WHILE-*program with an arbitrary number of registers. Then there is a* WHILE-*program P with register complexity three, such that $\varphi_P(n) = f(n)$, for all $n \in \mathbb{N}_0$.*

Proof. It is sufficient (1) to encode the original input x to 2^x and to decode the output 2^x to x, and (2) to simulate a two-counter machine with a WHILE-program using three registers. In order to show (2), let A be a two-counter machine with state set $S = \{s_1, s_2, \dots, s_n\}$, for some $n \geq 1$, and initial state s_1. The following while program P simulates A, where registers c_1 and c_2 are used to store the values of the counters. The third register is denoted by z. Initially, P starts with the input value stored in register c_1, and c_2 and z are set to 0.

```
$z := z + 1$;                                      {$z$ is set to 1 referring to $s_1$}
WHILE $z \neq 0$ DO
  $z := z - 1$; WHILE $z = 0$ DO $P_1$ OD;   {$P_1$ simulates state $s_1$}
      ⋮
  $z := z - 1$; WHILE $z = 0$ DO $P_n$ OD    {$P_n$ simulates state $s_n$}
OD
```

Apart from simulating state s_i, the sub-program P_i has to set the register z, such that the correct successor state of s_i is simulated during the next loop. The following sub-program P_i simulates state s_i. It starts with the current counter values stored in registers c_1 and c_2 and z are decremented to 0.

```
WHILE $c_1 = 0$ DO
    $z := z + 2$;              {if $c_1 = 0$, then $z := 2$}
    $c_1 := c_1 + 1$           {terminates the loop}
OD;
WHILE $c_2 = 0$ DO
    $z := z + 1$;              {if $c_2 = 0$, then $z := z + 1$}
    $c_2 := c_2 + 1$           {terminates the loop}
OD;
```

So far, P_i has performed the zero-tests. The value stored in z is interpreted as follows: $z = 0$ means $c_1 > 0$, $c_2 > 0$ and neither c_1 nor c_2 have been incremented in order to terminate the loop, $z = 1$ means $c_1 > 0$, $c_2 = 0$ and c_2 has been incremented and must be corrected, $z = 2$ means $c_1 = 0$, $c_2 > 0$ and c_1 has been incremented, and $z = 3$ means $c_1 = 0$, $c_2 = 0$ and both c_1 and c_2 have been incremented. Let $op \in \{+, n, -\}$ denote the possible manipulations of the counters c_m, $m \in \{1, 2\}$. For easier writing, $+(c_m)$ $(-(c_m))$ denotes $c_m := c_m + 1$ $(c_m := c_m - 1)$. The notion $n(c_m)$ is used for no operation. When the computation of the two-counter machine continues, we have $\delta(s_i, t_1, t_2) = (s_j, op_1, op_2)$, for some $1 \leq j \leq n$. Otherwise, we assume $\delta(s_i, t_1, t_2) = (s_0, n, n)$. Since for a state s_0 the main loop of program P terminates, P simulates the stop of A correctly. Sub-program P_i continues as follows:

```
WHILE $z = 0$ DO                        $\{c_1 > 0, c_2 > 0\}$
     $op_1(c_1)$;  $op_2(c_2)$;          $\{\delta(s_i, 1, 1) = (s_o, op_1, op_2)\}$
     $z := 3 + n - i + o$                $\{n$, $i$ and $o$ are constants$\}$
OD;
$z := z - 1$;
        ⋮
$z := z - 1$;
WHILE $z = 0$ DO                        $\{c_1 = 0, c_2 = 0\}$
     $c_1 := c_1 - 1$;                   $\{$correction of $c_1\}$
     $c_2 := c_2 - 1$;                   $\{$correction of $c_2\}$
     $op_1(c_1)$;  $op_2(c_2)$;          $\{\delta(s_i, 0, 0) = (s_r, op_1, op_2)\}$
     $z := n - i + r$                    $\{n$, $i$ and $r$ are constants$\}$
OD
```

□

Corollary 6. WHILE = WHILE$_3$, *there is a universal while program with three registers.*

In order to separate one from two registers in case of WHILE-programs, we have the following characterization of total functions that are computable by WHILE-programs with one register.

Lemma 7. *Let P be a* WHILE*-program with one register. If φ_P is total, then there are constants $n_0, c \in \mathbb{N}_0$ and $d \in \mathbb{Z}$, such that for all $n \geq n_0$ either $\varphi_P(n) = c$ or $\varphi_P(n) = n + d$.*

Proof. Let P be a WHILE-program with one register x such that φ_P is total. We observe that each WHILE-command of the form WHILE $x \neq 0$ DO P_1 OD can be replaced by the assignment $x := 0$. This can be done since either x is 0 and the WHILE-command is not executed, or $x \neq 0$ and the WHILE-command is executed until x is 0. We obtain an equivalent program P'.

If in the command sequence $P_1; \ldots; P_m$ of P' there appears an assignment $x := 0$, say as P_i, then we can delete the commands P_1 to P_{i-1} safely without changing $\varphi_{P'}$. This transformation is repeated until at most one assignment $x := 0$ is left. Again we obtain an equivalent program P'', which has the form $P_1; \ldots; P_r$, where the commands P_i are either $x := x + 1$, $x := x - 1$, or a statement WHILE $x = 0$ DO P_i' OD. The only exception is P_1 which may also be $x := 0$. Set $d = u - v$, where u is the number of commands $x := x + 1$ and v is the number of commands $x := x - 1$ appearing in the sequence $P_1; \ldots; P_r$.

If P_1 is $x := 0$, then P'' computes the result $c = \varphi_{P''}(0)$ for all inputs. Otherwise let $n_0 = r + 1$. For any input $n \geq n_0$ none of the WHILE-commands in P'' is executed, since there are only r commands and, thus, x is decremented at most r times, but the condition is always $x = 0$. So, for any $n \geq n_0$ the program computes the result $\varphi_{P''}(n) = n + d$. □

Theorem 8. WHILE$_1$ ⊂ WHILE$_2$.

Proof. Consider the total function $f(n) = 2n$ as witness. □

4.2 GOTO-Programs

Theorem 9. *Let $k \in \mathbb{N}$ be a constant. A function $f : \mathbb{N}_0 \hookrightarrow \mathbb{N}_0$ is computable by a* `GOTO`*-program with k registers if and only if it is computable by a k-counter machine.*

Proof. Given some `GOTO`-program P with k registers, the first step of a construction of an equivalent k-counter machine A is to replace every command of the form $x := 0$ by $M_i : x := x - 1$; M_{i+1} : `IF` $x \neq 0$ `GOTO` M_i, where M_i, M_{i+1} are new labels. The transformed program P' is equivalent to P. Without loss of generality, we may assume that every command in P' is labeled, and that the labels are $M_1, \dots, M_r$ in order of occurence. So, program P' has flow chart form $M_1 : P_1$; $M_2 : P_2$; $\dots M_r : P_r$, where P_i, $1 \leq i \leq r$ is a command. Let the registers be named $c_1, \dots, c_k$. Automaton A has one state for every labeled command, i.e., the state set is $S = \{s_1, \dots s_r\}$. Accordingly, the initial state is s_1. The transition function δ is for $t_1, \dots, t_k \in \{0, 1\}$, $s_i, s_{i+1}, s_p \in S$, and $1 \leq j \leq k$ defined as follows:

$$\delta(s_i, t_1, \dots, t_k) = \begin{cases} (s_{i+1}, n^{j-1} + n^{k-j}) & \text{if } P_i \equiv c_j := c_j + 1 \\ (s_{i+1}, n^{j-1} - n^{k-j}) & \text{if } P_i \equiv c_j := c_j - 1 \\ (s_p, n^k) & \text{if } P_i \equiv \texttt{GOTO}\ M_p \\ (s_p, n^k) & \text{if } t_j = 0 \text{ and } P_i \equiv \texttt{IF}\ x_j = 0\ \texttt{GOTO}\ M_p \\ (s_{i+1}, n^k) & \text{if } t_j \neq 0 \text{ and } P_i \equiv \texttt{IF}\ x_j = 0\ \texttt{GOTO}\ M_p \\ (s_p, n^k) & \text{if } t_j \neq 0 \text{ and } P_i \equiv \texttt{IF}\ x_j \neq 0\ \texttt{GOTO}\ M_p \\ (s_{i+1}, n^k) & \text{if } t_j = 0 \text{ and } P_i \equiv \texttt{IF}\ x_j \neq 0\ \texttt{GOTO}\ M_p \end{cases}$$

Conversely, let A be a k-counter machine. The construction of an equivalent `GOTO`-program with k registers is shown for the case $k = 2$. The generalization to arbitrary k is straightforward. Let $S = \{s_1, s_2, \dots, s_n\}$, for some $n \geq 1$, be the state set of A, whose initial state is s_1. The following `GOTO`-program P simulates A, where registers c_1 and c_2 are used to store the values of the counters. Initially, P starts with the input value stored in register c_1, and c_2 is set to 0.

```
M_1:  P_1;                          {P_1 simulates state s_1}
      ⋮
M_n:  P_n;                          {P_n simulates state s_n}
M_0:                                {no operation to terminate P}
```

The sub-programs P_i are as follows, where we use the notions as in the proof of Theorem 5.

```
      IF c_1 = 0 GOTO N_{i,0};
      IF c_2 = 0 GOTO N_{i,1,0};
      op_1(c_1);  op_2(c_2);        {c_1 > 0, c_2 > 0}
      GOTO M_o;                     {δ(s_i, 1, 1) = (s_o, op_1, op_2)}
```

```
N_{i,0}:   IF $c_2 = 0$ GOTO $N_{i,0,0}$;
           $op_1(c_1)$; $op_2(c_2)$;          $\{c_1 = 0, c_2 > 0\}$
           GOTO $M_q$;                        $\{\delta(s_i, 0, 1) = (s_q, op_1, op_2)\}$
$N_{i,1,0}$: $op_1(c_1)$; $op_2(c_2)$;        $\{c_1 > 0, c_2 = 0\}$
           GOTO $M_p$;                        $\{\delta(s_i, 1, 0) = (s_p, op_1, op_2)\}$
$N_{i,0,0}$: $op_1(c_1)$; $op_2(c_2)$;        $\{c_1 = 0, c_2 = 0\}$
           GOTO $M_r$                         $\{\delta(s_i, 0, 0) = (s_r, op_1, op_2)\}$
```

□

Corollary 10. $\mathtt{GOTO} = \mathtt{GOTO}_3$*, there is a universal* GOTO*-program with three registers.*

The witness function $f(n) = 2n$ separates the first two levels:

Theorem 11. $\mathtt{GOTO}_1 \subset \mathtt{GOTO}_2$

5 Conclusions

We have done a few steps towards the exploration of register complexity of LOOP-, WHILE-, and GOTO-programs. There are still many open questions and interesting problems. Finally, we briefly discuss some of them.

- Are the inclusions $\mathtt{LOOP}_2 \subseteq \mathtt{LOOP}_3 \subseteq \mathtt{LOOP}_4$, $\mathtt{WHILE}_2 \subseteq \mathtt{WHILE}_3$, or $\mathtt{GOTO}_2 \subseteq \mathtt{GOTO}_3$ strict?

Since the nesting depth of LOOP-programs with a fixed number of registers can be arbitrarily large, it seems to be difficult to argue with upper or lower bounds on the running time or on the sum of register values in order to separate the levels. On one hand, the technique used to separate the first level, i.e., to characterize the computable functions and to identify gaps in the orders of magnitude, might be refined for higher levels. On the other hand, the gaps could be too small for this technique.

- Which sets of register operations are universal, i.e., allow to construct universal programs? For historical reasons, we have incrementation, decrementation, and initialization with zero, but, in general, one can ask for any reasonable set of operations.

From our set, $x := 0$ can safely be replaced. On the other hand, we cannot omit the incrementation (from our set), since otherwise, e.g., non-input registers would always keep the value zero. What about decrementation? For LOOP-programs, $x := x - 1$ can be replaced by $y := 0; z := 0;$ LOOP x DO LOOP z DO $y := y + 1$ OD; $z := 0; z := z + 1$ OD followed by $x := 0;$ LOOP y DO $x := x + 1$ OD. Since the construction takes extra registers, the register complexity seems to be sensitive to the set of operations, what raises the next questions.

- What are the relations between the number of necessary registers and the set of operations?

Similarly, one may ask for the role played by the allowed conditions. Here we have tests $= 0$ and $\neq 0$. If we allow more general tests, e.g., $x = y$ or $x \neq y$, we can omit the decrementation $x := x - 1$ from `WHILE`- and `GOTO`-programs. But again, this takes extra registers. Conversely, one can omit the test $= 0$ at the cost of extra registers. So, the set of operations and the set of tests and the number of registers are sensitively connected.

- What are the relations between the number of necessary registers and the set of tests?

Acknowledgements

The authors are grateful to Rudolf Freund who pointed out an error in an earlier version of the paper.

References

1. A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363, 1936.
2. E. Engler and P. Läuchli. *Berechnungstheorie für Informatiker*. Teubner, 1988.
3. R. Freund, C. Martín-Vide, G. Păun. From regulated rewriting to computing with membranes: collapsing hierarchies. *Theoretical Computer Science*, 312:143–188, 2004.
4. A. Grzegorczyk. Some classes of recursive functions. *Rozprawy Matematycne*, 4:1–45, 1953.
5. A. J. Kfoury, R. N. Moll, and M. A. Arbib. *A Programming Approach to Computability*. Texts and Monographs in Computer Science. Springer, 1982.
6. S. C. Kleene. General recursive functions of natural numbers. *Mathematische Annalen*, 112:727–742, 1936.
7. A. R. Meyer and D. M. Ritchie. The complexity of loop programs. In *Proceedings of the ACM National Meeting*, pages 465–469. American Mathematical Society, 1967.
8. M. L. Minsky. *Computation: Finite and Infinite Machines*. Automatic Computation. Prentice-Hall, 1967.
9. E. L. Post. Finite combinatory processes—formulation 1. *The Journal of Symbolic Logic*, 1:103–105, 1936.
10. J. C. Shepherdson and H. E. Sturgis. Computability of recursive functions. *Journal of the ACM*, 10:217–255, 1963.
11. A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936.
12. A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. A correction. *Proceedings of the London Mathematical Society*, 43:544–546, 1937.

Classification and Universality of Reversible Logic Elements with One-Bit Memory

Kenichi Morita, Tsuyoshi Ogiro, Keiji Tanaka, and Hiroko Kato

Hiroshima University, Graduate School of Engineering,
Higashi-Hiroshima, 739-8527, Japan
morita@iec.hiroshima-u.ac.jp

Abstract. A reversible logic element is a model of a computing element that has an analogous property to physical reversibility. In this paper, we investigate k-symbol reversible elements with one-bit memory (i.e., two states) for $k = 2$, 3, and 4. We classified all of them, and showed the total numbers of essentially different elements are 8 ($k = 2$), 24 ($k = 3$), and 82 ($k = 4$). So far, a rotary element, a 2-state 4-symbol reversible element, has been known to be logically universal. Here, we proved that a new and interesting elements called 3- and 4-symbol left-/right-rotate elements are both universal. We also gave a new concise construction method of a Fredkin gate out of rotary elements.

Keywords: reversible logic element, logical universality, rotary element, Fredkin gate

1 Introduction

Reversible computing (see e.g. [1,2]) is a paradigm of computing that has a property analogous to physical reversibility, and is related to quantum computing (see e.g. [4]). So far, reversible Turing machines, reversible cellular automata, etc. have been proposed and investigated as models of reversible computing.

Reversible logic elements and circuits were first studied by Toffoli [8,9] and Fredkin and Toffoli [3]. There are "universal" reversible elements in the sense that any logic function (even if it is irreversible) can be realized in a circuit composed only of them. A Fredkin gate [3] and a Toffoli gate [8] are typical universal reversible gates having 3 inputs and 3 outputs. Hence a universal computer can be constructed as a circuit composed only of such gates and delay elements.

Besides reversible gates (i.e., elements without memory), there are also universal reversible elements with memory. A rotary element [6,7] is a one having one-bit memory (i.e., two states) and 4 input/output symbols (or it can be regarded as having 4 input/output lines). It was shown that a reversible element with memory like RE is very useful when constructing reversible computers such as reversible Turing machines and counter machines concisely [6,7].

In this paper, we investigate 2-state k-symbol reversible logic elements for $k = 2$, 3, and 4 to find logically universal elements. We classify all of them, and show the total numbers of equivalence classes of elements are 8 ($k = 2$), 24 ($k = 3$), and 82 ($k = 4$). Here, we introduce a new and interesting elements called 3- and 4-symbol left-/right-rotate elements (3LRRE and 4LRRE), and proved their logical universality. We also give a new construction method of a Fredkin gate out of rotary elements, which improves the method given in [7].

2 Reversible Logic Elements

2.1 Logic Elements Formalized as Reversible Sequential Machines

Since a reversible logic element with memory can be formalized as a reversible sequential machine (RSM), we first give a definition of the latter.

A *sequential machine* (SM) (of Mealy type) is a system defined by

$$M = (Q, \Sigma, \Gamma, q_0, \delta),$$

where Q is a finite non-empty set of states, Σ and Γ are finite non-empty sets of input and output symbols, respectively, and $q_0 \in Q$ is an initial state. $\delta : Q \times \Sigma \to Q \times \Gamma$ is a mapping called a *move function*. A variation of an SM $M = (Q, \Sigma, \Gamma, \delta)$, where no initial state is specified, is also called an SM.

M is called a *reversible sequential machine* (RSM) if δ is one-to-one (hence $|\Sigma| \le |\Gamma|$). An RSM is "reversible" in the sense that, from the present state and the output of M, the previous state and the input are determined uniquely.

In the rest of this paper, we mainly investigate RSMs with $|Q| = 2$ and $|\Sigma| = |\Gamma| = k$. Here, such an RSM is also called a 2*-state k-symbol reversible logic element*, because we investigate how universal computers can be built by using only such elements.

2.2 Rotary Element (RE): An Example of a Reversible Logic Element

A *rotary element* (RE) [6,7] is a 2-state 4-symbol reversible element defined by

$$\mathrm{RE} = (\{\boxminus, \boxed{|}\}, \{n, e, s, w\}, \{n', e', s', w'\}, \delta_{\mathrm{RE}}),$$

where the move function δ_{RE} is shown in Table 1 (for instance, if the present state is $\boxminus$ and the input is n, then the state becomes $\boxed{|}$ and the output is w'). It is easily verified that an RE is reversible.

Present state	Input			
	n	e	s	w
H-state: $\boxminus$	$\boxed{|}$ w'	$\boxminus$ w'	$\boxed{|}$ e'	$\boxminus$ e'
V-state: $\boxed{|}$	$\boxed{|}$ s'	$\boxminus$ n'	$\boxed{|}$ n'	$\boxminus$ s'

Table 1. The move function δ_{RE} of a rotary element RE.

The operation of an RE can be understood by the following intuitive interpretation. It has two states called H-state ($\boxminus$) and V-state ($\boxed{|}$), and four input lines $\{n, e, s, w\}$ and four output lines $\{n', e', s', w'\}$ corresponding to the input and output alphabets. All the values of input and output lines are either 0 or 1, i.e., $(n, e, s, w), (n', e', s', w') \in \{0, 1\}^4$. However, we restrict the domain of their values as $\{(1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1)\}$, i.e., exactly one "1" appears among the four input (output, respectively) lines at a time, when an input is given (an output is produced). We also assume if all the values of input lines are 0's then neither state-transition nor output-production occurs. The operation of an RE is left undefined for the cases that signal 1's are given to two or more input lines. Signals 1 and 0 are interpreted as existence and non-existence of a particle. We can regard an RE has a "rotating bar" to control the moving direction of a particle. When no particle exists, nothing happens on the RE. If a particle comes from a direction parallel to the rotating bar, then it goes out from the output line of the opposite side (i.e., it goes straight ahead) without affecting the direction of the bar (Fig. 1 (a)). On the other hand, if a particle comes from a direction orthogonal to the bar, then it makes a right turn, and rotates the bar by 90 degrees counterclockwise (Fig. 1 (b)).

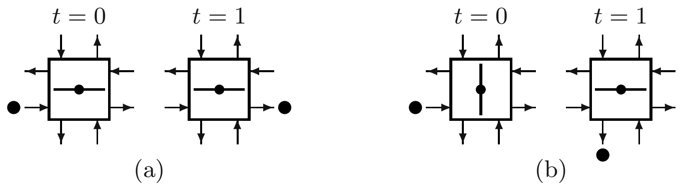


Fig. 1. Operations of an RE: (a) the parallel case (i.e., the coming direction of a particle is parallel to the rotating bar), and (b) the orthogonal case.

Based on such an interpretation, we define a *reversible logic circuit*. It is a circuit composed only of reversible logic elements satisfying the following condition. Each output of an element can be connected at most one input of some other (or may be the same) element, i.e., "fan-out" of an output is not allowed. This condition corresponds to the conservation law in physics [3].

In the following sections, we use a pictorial representation of a move function of a 2-state reversible element instead of using a table. Fig. 2 shows such a representation of the move function of an RE.

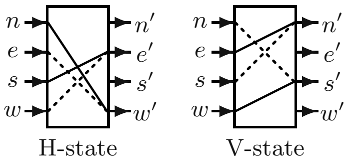


Fig. 2. Pictorial representation of the move function of an RE. Solid and dotted lines in a box describe the input-output relation for each state. Further, a solid line shows the state transits to another, and a dotted line shows the state remains unchanged.

2.3 Logical Universality of Reversible Logic Elements

In this paper, we define the notion of logical universality as follows. A set of logic elements is called *logically universal* if any sequential machine can be built as a circuit composed only of the elements in the set (supposing some appropriate "coding/decoding" method between state/symbols of the SM and those of the logic elements). It is very well known that the traditional set of logic elements {AND, NOT, delay element} is logcally universal.

A Fredkin gate and a Toffoli gate are 3-input 3-output reversible logic gates that are known to be logically universal. (From the viewpoint of our formalism, they can be regarded as 1-state 8-symbol reversible logic elements if unit-delays are attached to them.) A Fredkin gate [3] is an element realizing the mapping $(c, p, q) \mapsto (c, cp + \bar{c}q, cq + \bar{c}p)$. Since AND and NOT can be constructed from Fredkin gate [3], the set {Fredkin gate, delay element} is logically universal. A Toffoli gate [8] realizes $(x, y, z) \mapsto (x, y, (xy) \oplus z)$. Also in this case AND and NOT can be constructed from Toffoli gate. Hence, {Toffoli gate, delay element} is logically universal.

Among 2-state reversible logic elements, an RE is the first one that was shown to be logically universal. In [7] a construction method of a Fredkin gate from 16 REs and delay elements was shown. It is also shown that any reversible Turing machine can be composed only of REs very concisely [7]. Here we give an improved result in Fig. 3 where a Fredkin gate is realized by using 8 REs and delay elements. Since an RE can also operate as a simple delay element as in Fig.1(a), we can conclude the set {RE} is logically universal.

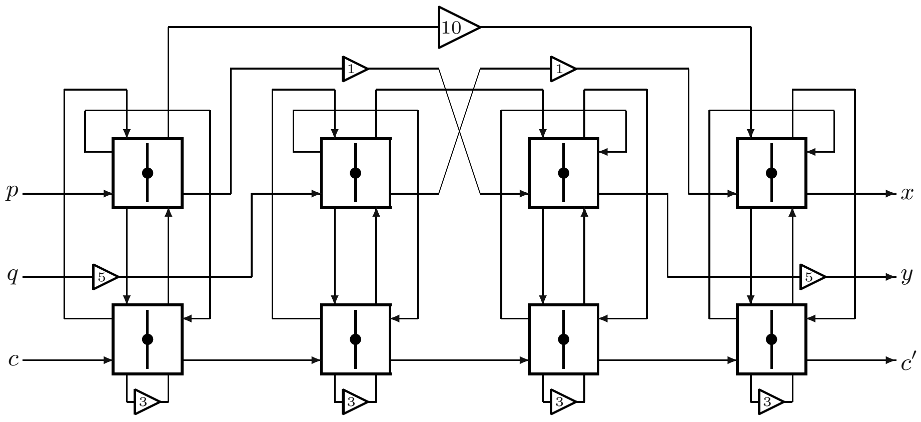


Fig. 3. A new realization method of a Fredkin gate out of 8 REs and delay elements, where $c' = c$, $x = cp + \bar{c}q$, $y = cq + \bar{c}p$. Small triangles are delay elements, where the number written inside of each triangle indicates its delay time. The total delay time between inputs and outputs is 20.

3 Clasification of 2-State Reversible Elements

In this section, we investigate 2-state k-symbol reversible logic elements for $k = 2, 3$, and 4, and show how many elements exist and how they are classified.

3.1 Equivalence Classes of 2-State Reversible Elements

Consider a 2-state 4-symbol reversible element $M = (Q, \Sigma, \Gamma, \delta)$. We fix the state set as $Q = \{q_0, q_1\}$, and the input and output alphabets as $\Sigma = \{a, b, c, d\}$ and $\Gamma = \{w, x, y, z\}$, respectively. Then the move function δ is as follows.

$$\delta : \{q_0, q_1\} \times \{a, b, c, d\} \to \{q_0, q_1\} \times \{w, x, y, z\}$$

Since δ must be one-to-one, it is specified by a permutation from the set

$$\{(q_0, w), (q_0, x), (q_0, y), (q_0, z), (q_1, w), (q_1, x), (q_1, y), (q_1, z)\}.$$

Hence, there are $8! = 40320$ elements in total. They are numbered by $0, \cdots, 40319$ in the lexicographic order of permutations. 2-state 2- and 3-symbol reversible elements are also numbered in a similar manner. To indicate k-symbol element, the prefix "k-" is attached to the serial number. Table 2 shows two examples of elements No. 4-289 and No. 4-37963 specified by the following permutations (in Table. 2(b), each symbol have a prime ($'$) for the sake of the later argument).

No. 4-289: $((q_0, w), (q_0, x), (q_1, w), (q_1, x), (q_0, y), (q_0, z), (q_1, z), (q_1, y))$
No. 4-37963: $((q_1, z), (q_0, z), (q_1, x), (q_0, x), (q_1, y), (q_0, w), (q_1, w), (q_0, y))$

We can regard two elements are "equivalent" if one can be obtained by renaming the states and the input/output symbols of the other. This notion is formalized as follows.

Let $M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1)$ and $M_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2)$ be two SMs. M_1 and M_2 are called *equivalent* (denoted by $M_1 \sim M_2$), if there exist one-to-one onto mappings (bijections) $f : Q_1 \to Q_2$, $g : \Sigma_1 \to \Sigma_2$, and $h : \Gamma_1 \to \Gamma_2$ that satisfy

$$\forall q \in Q_1,\ \forall s \in \Sigma_1\ [\,\delta_1(q, s) = \psi(\delta_2(f(q), g(s)))\,],$$

where $\psi : Q_2 \times \Gamma_2 \to Q_1 \times \Gamma_1$ is defined as follows.

$$\forall q \in Q_2,\ \forall t \in \Gamma_2\ [\,\psi(q, t) = (f^{-1}(q), h^{-1}(t))\,]$$

Two elements No. 4-289 and No. 4-37963 are equivalent under the following bijections (they are in fact equivalent to an RE).

$$\begin{aligned}&f(q_0) = q_0',\ f(q_1) = q_1'\\&g(a) = b',\ \ g(b) = d',\ \ g(c) = a',\ \ g(d) = c'\\&h(w) = z',\ h(x) = x',\ h(y) = w',\ h(z) = y'\end{aligned}$$

The total numbers of 2-state 2-, 3-, and 4-symbol reversible elements are $4! = 24$, $6! = 720$, and $8! = 40320$, respectively. We made a computer program that computes all equivalence classes of them. The result is given in Figs. 4–6. These figures show all representative elements in the equivalence classes of 2-, 3-, and 4-symbol reversible elements. The representatives are so chosen that it has the smallest index number in the equivalence class.

Present state	Input			
	a	b	c	d
q_0	$q_0 w$	$q_0 x$	$q_1 w$	$q_1 x$
q_1	$q_0 y$	$q_0 z$	$q_1 z$	$q_1 y$

(a) Element No. 4-289

Present state	Input			
	a'	b'	c'	d'
q'_0	$q'_1 z'$	$q'_0 z'$	$q'_1 x'$	$q'_0 x'$
q'_1	$q'_1 y'$	$q'_0 w'$	$q'_1 w'$	$q'_0 y'$

(b) Element No. 4-37963

Table 2. An example of a pair of equivalent 2-state 4-symbol reversible elements.

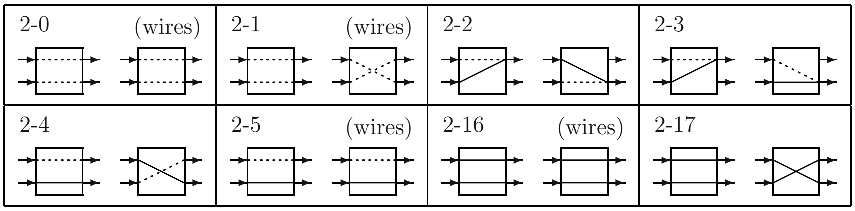


Fig. 4. Representatives of 8 equivalence classes of 24 2-state 2-symbol reversible elements.

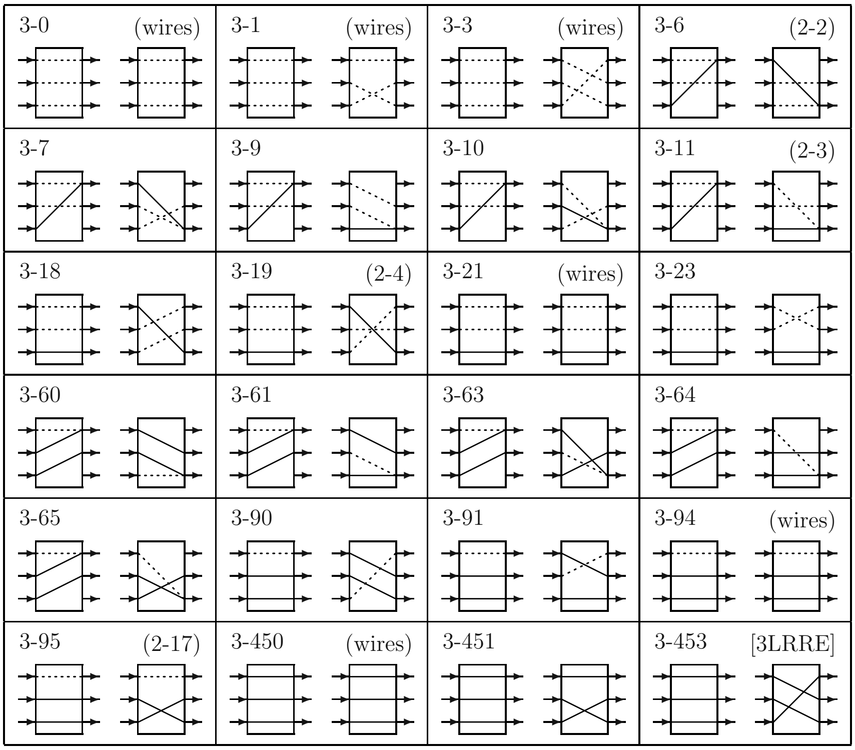


Fig. 5. Representatives of 24 equivalence classes of 720 2-state 3-symbol reversible elements.

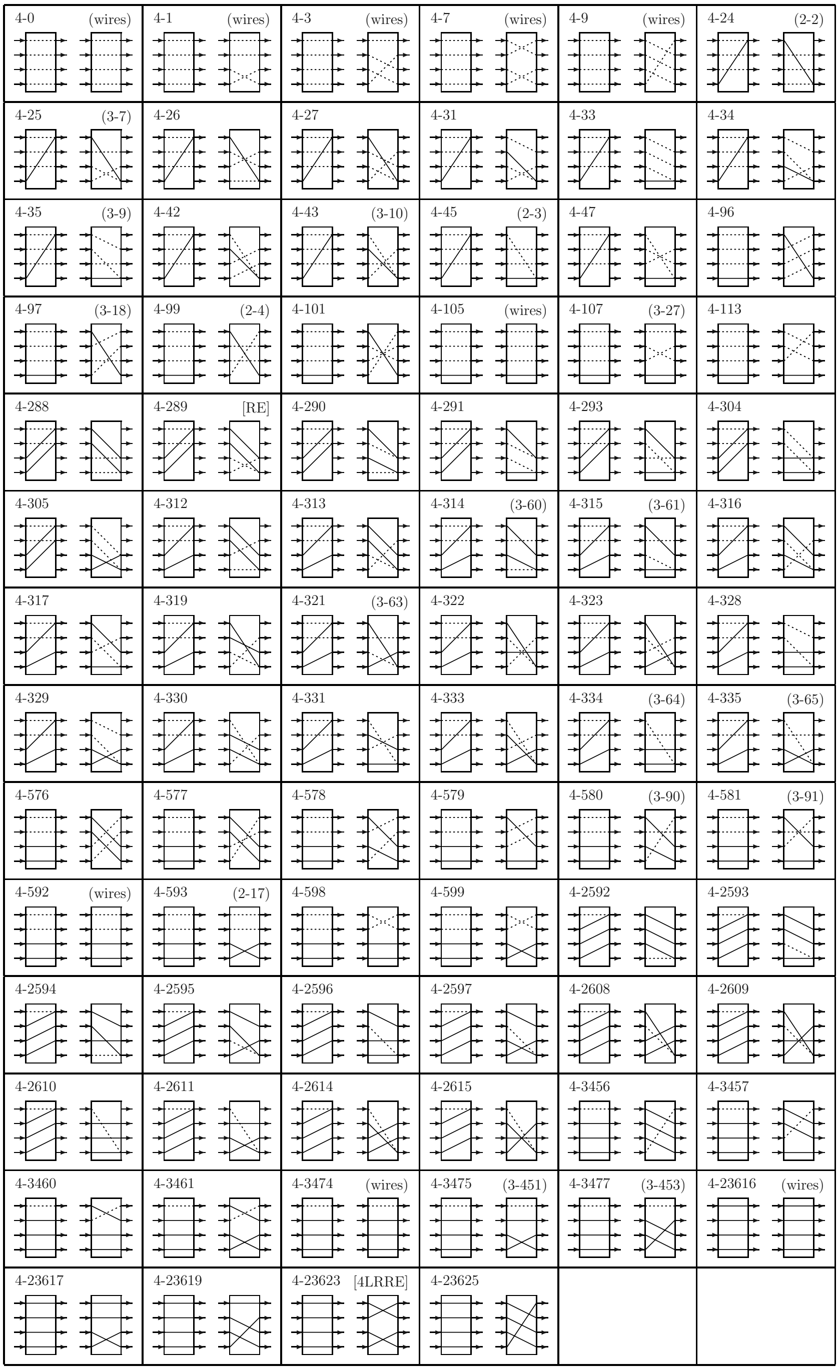


Fig. 6. Representatives of 82 equivalence classes of 40320 2-state 4-symbol reversible elements.

3.2 Classifying Representative Elements into Three Categories

As shown in Figs. 4–6 total numbers of equivalence classes of k-symbol reversible elements for $k = 2$, 3, and 4 are 8, 24, and 82, respectively. We further classify them into the following three categories.

1. Elements equivalent to connecting wires:
   This is a degenerate case. For example, the element No. 3-0 in Fig. 5 acts as three wires that simply connect input and output lines. (Note that such a wire has a unit-time delay. Therefore, it can be regarded as a 1-state 1-symbol RSM.) No. 3-1 is also the case, because no input makes state-change. Apparently, such elements have no logical function.
2. Elements equivalent to a simpler element with fewer symbols:
   This is also a degenerate case, and reducible to the elements with fewer input/output symbols. For example, the element No. 3-6 is equivalent to the element No. 2-2 plus a simple wire.
3. Proper k-symbol elements:
   All the elements other than the cases 1 and 2 are "proper" k-symbol elements. This is a non-degenerate case. The elements in this category are the main concern of the investigation for each k.

Classification can be done by checking the move function (or its pictorial representation) of each representative element. In Figs. 4–6, the cases 1 and 2 are indicated at the upper-right corner of each box. Table 3 shows the number of elements in each category for k =2, 3, and 4.

	Total number of representative elements	1. Elements equivalent to connecting wires	2. Elements equivalent to simpler elements	3. Nondegenerate k-symbol elements
$k = 2$	8	4	0	4
$k = 3$	24	6	4	14
$k = 4$	82	9	18	55

Table 3. Classification of representatives of 2-state k-symbol reversible elements.

4 Left-/Right-Rotate Elements and Their Universality

In this section, we investigate the specific reversible elements No. 3-598 and No. 4-29514 called left-/right-rotate elements with 3- and 4-input/output lines (3LRRE and 4LRRE), which are equivalent to the representatives No. 3-453 and No.4-23623, respectively. The operations of them are very simple, mainly because they are isotropic (explained later). In spite of their simplicity, we can show logical universality of them.

4.1 Left-/Right-Rotate Element with 4-Input/Output Lines (4LRRE)

The element No. 4-29514 shown in Fig. 7 is called a *left-/right-rotate element with 4-input/output lines* (4LRRE). If we depict a 4LRRE by a square-shaped one shown in Fig. 8, its operation is easily understood. That is, L-state (R-state, respectively) makes every coming signal turn left (right), and then makes the state transit to R-state (L-state).

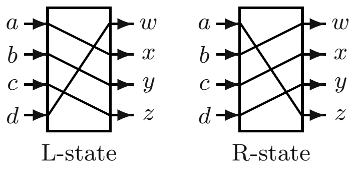


Fig. 7. Element No. 4-29514 called a 4LRRE (equivalent to the representative element No. 4-23623).

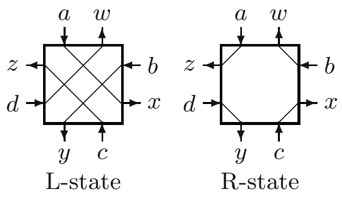


Fig. 8. Depicting a 4LRRE as a square-shaped element.

A 4LRRE is "isotropic" in the following sense. A 4-symbol reversible element $M = (Q, \Sigma, \Gamma, \delta)$ is called *isotropic* (or *rotation symmetric*) if the following condition holds.

$$\forall q, q' \in Q,\ s \in \Sigma,\ t \in \Gamma\ [\text{ if } \delta(q,s) = (q',t) \text{ then } \delta(q,\pi(s)) = (q',\pi(t))\ ],$$

where $\pi : \Sigma \cup \Gamma \to \Sigma \cup \Gamma$ is the following permutation.

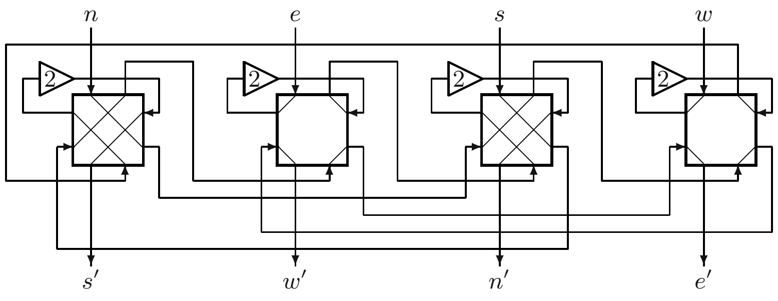
$$\pi(a) = b,\ \ \pi(b) = c,\ \ \pi(c) = d,\ \pi(d) = a,$$
$$\pi(w) = x,\ \pi(x) = y,\ \pi(y) = z,\ \pi(z) = w.$$

This notion can be generalized to k-symbol reversible elements by defining the permutation π similarly as above.

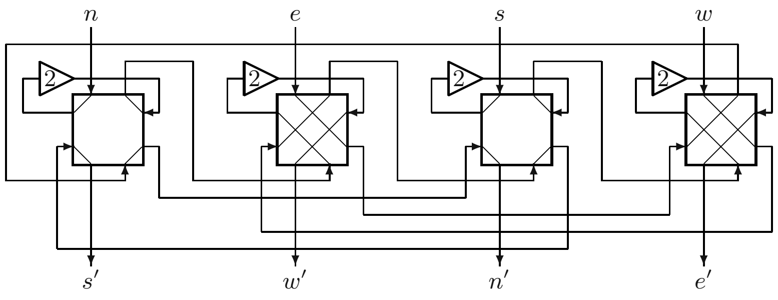
We now show that the universality of a 4LRRE. An RE is simulated by a circuit composed of 4LRREs and delay elements as in Fig. 9. H-state and V-state of an RE correspond to Fig. 9 (a) and (b), respectively. If a signal comes from the input line n of Fig. 9 (a), then the signal passes through all the four 4LRREs, and goes out from w'. Hence the state of the circuit becomes as in Fig. 9 (b).

On the other hand, if a signal comes from the input line n of Fig. 9 (b), then the signal passes through only the leftmost 4LRRE twice, and goes out from s'. Hence the state of the circuit remains unchanged. The other cases are similar to the above.

A delay of two units of time can be realized as shown in Fig. 10 by using a 4LRRE. (If we do so, two units of time should be regarded as a new unit of time hereafter.) Note that the state of the 4LRRE will be restored to the R-state after the operation, hence the delay circuit can be used as a delay element repeatedly. By above, we can conclude that the set {4LRRE} is logically universal.



(a) An RE of H-state.



(b) An RE of V-state.

Fig. 9. Realization of an RE by 4LRREs. The total time of delay between input and output is 4.

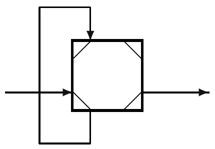


Fig. 10. Realization of a delay of two units of time by a 4LRRE.

4.2 Left-/Right-Rotate Element with 3-Input/Output Lines (3LRRE)

The element No. 3-598 shown in Fig. 11 is called a *left-/right-rotate element with 3-input/output lines* (3LRRE). (It is also possible to depict 3LRRE as a triangular-shaped element that makes a coming signal turn left or right depending on its state.)

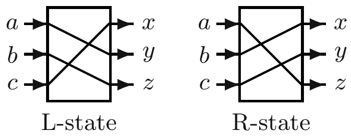


Fig. 11. Element No. 3-598 called a 3LRRE (equivalent to the representative element No. 3-453).

Although a 3LRRE is a very simple element, we can construct a circuit that simulates a Fredkin gate by using 3LRREs and delay elements as shown in Fig. 12. The initial states of the 3LRREs are all L-states, and after an operation all the states are restored to the L-states. It is also easy to make a delay of two units of time by a 3LRRE as in 4LRRE. Hence, we can conclude that the set {3LRRE} is logically universal.

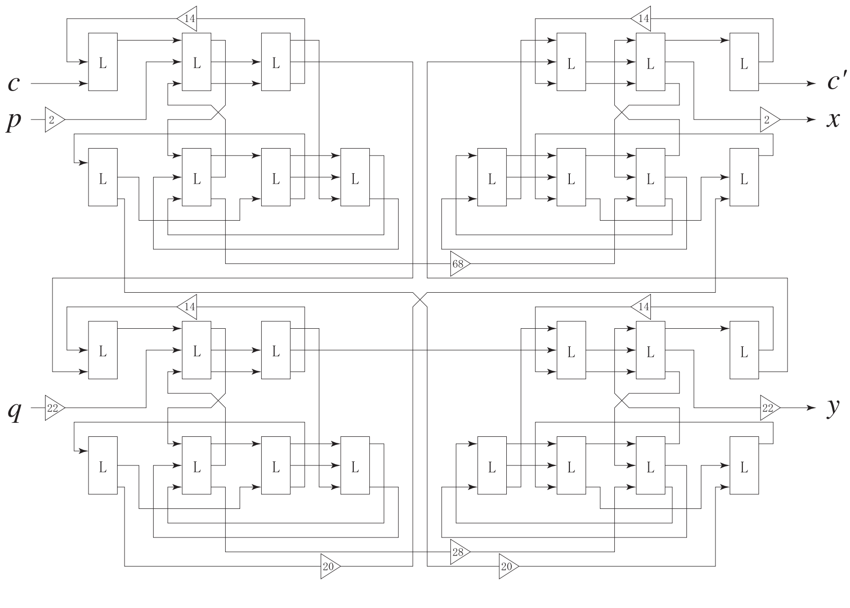


Fig. 12. A realization method of a Fredkin gate out of 3LRREs and delay elements, where $c' = c,\ x = cp + \bar{c}q,\ y = cq + \bar{c}p$. Initially, all the 3LRREs are set to L-states. The total delay time between inputs and outputs is 80.

5 Concluding Remarks

In this paper, we investigated and classified 2-state k-symbol reversible logic elements for $k = 2$, 3, and 4. We introduced specific simple 3- and 4-symbol elements called 3LRRE and 4LRRE, and showed they are both logically universal. In the cases of $k = 3$ and 4, it is likely that there are many simple and interesting reversible elements with logical universality other than RE, 3LRRE, and 4LRRE. In the case of 2-state 2-symbol RLEMs, Lee et al. [5] showed that the set {No.2-3, No.2-4} is logically universal, i.e., a Fredkin gate can be constructed by using both two RLEMs No.2-3 and No.2-4. On the other hand, we conjecture each of all the 2-state 2-symbol reversible elementsare non-universal. These problems are left for the future study.

Acknowledgement. The authors would express their thanks to Dr. Chuzo Iwamoto and Dr. Katsunobu Imai of Hiroshima University for their helpful discussions and useful comments. One of the author K. Morita is also grateful to Dr. Ferdinand Peper and the members of his project in Kansai Advanced Research Center of NICT. This work was supported in part by Grant-in-Aid for Scientific Research (C) No. 16500012 from JSPS.

References

1. Bennett, C.H., Logical reversibility of computation, *IBM J. Res. Dev.*, **17**, 525–532 (1973).
2. Bennett, C.H., Notes on the history of reversible computation, *IBM J. Res. Dev.*, **32**, 16–23 (1988).
3. Fredkin, E. and Toffoli, T., Conservative logic, *Int. J. Theoret. Phys.*, **21**, 219–253 (1982).
4. Gruska, J., *Quantum Computing*, McGraw-Hill, London (1999).
5. Lee, J., Peper, F., Adachi, S. and Mashiko, S., Asynchronously timed reversible logic elements, (submitted for publication).
6. Morita, K., Tojima, Y. and Imai, K., A simple computer embedded in a reversible and number-conserving two-dimensional cellular space, *Multiple-Valued Logic*, **6**, 483–514 (2001).
7. Morita, K., A simple universal logic element and cellular automata for reversible computing, *Proc. 3rd Int. Conference on Machines, Computations, and Universality*, Chisinau, LNCS 2055, Springer-Verlag, 102–113 (2001).
8. Toffoli, T., Reversible computing, in *Automata, Languages and Programming*, Springer-Verlag, LNCS-85, 632–644 (1980).
9. Toffoli, T., Bicontinuous extensions of invertible combinatorial functions, *Mathematical Systems Theory*, **14**, 12–23 (1981).

Universal Families of Reversible P Systems

Alberto Leporati, Claudio Zandron, and Giancarlo Mauri

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano – Bicocca
Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
{leporati, zandron, mauri}@disco.unimib.it

Abstract. Conservative logic is a mathematical model of computation that reflects some properties of microdynamical laws of physics, such as reversibility and the conservation of the internal energy of the physical system used to perform the computations. The model is based upon the Fredkin gate, a reversible and conservative three–input/three–output boolean gate which is functionally complete for boolean logic. Computations are performed by reversible circuits composed by Fredkin gates.
In this paper we introduce energy–based P systems as a parallel and distributed model of computation in which the amount of energy manipulated and/or consumed during computations is taken into account. Moreover, we show how energy–based P systems can be used to simulate reversible Fredkin circuits. The simulating systems turn out to be themselves reversible and conservative.

1 Introduction

Reversibility plays a fundamental role when the possibility to perform computations with minimal energy dissipation is considered. Studies on thermodynamics of computing started in the 1950's, and continued in the following decades. As a result some bounds on the amount of dissipated energy during transmission and computation were established [11,3,2,12]. As shown in [11], erasing a bit necessarily dissipates $kT \ln 2$ Joule in a computer operating at temperature T, and generates a corresponding amount of entropy. Here k is Boltzmann's constant and T the absolute temperature in degrees Kelvin, so that $kT \approx 3 \times 10^{-21}$ Joule at room temperature. However, in [11] Landauer also demonstrated that only *logically irreversible* operations necessarily dissipate energy when performed by a physical computer. (An operation is *logically reversible* if its inputs can always be deduced from its outputs.) This result gave substance to the idea that logically reversible computations could be performed with zero internal energy dissipation. Indeed, since the appearance of [11] many authors have concentrated their attention on reversible computations. The importance of reversibility has grown further with the development of *quantum computing*, where the dynamical behavior of quantum systems is usually described by means of unitary operators, which are inherently logically reversible.

Many papers on reversible computation have appeared in literature, the most famous of which is certainly the work of Bennett on (universal) reversible Turing machines [3]. Here we consider the work of Fredkin and Toffoli on conservative logic [7], which is a mathematical model that allows one to describe computations which reflect some properties of microdynamical laws of physics, such as reversibility and conservation of the internal energy of the physical system used to perform the computations. In this model, computations are performed by reversible circuits composed by Fredkin gates.

In this paper we introduce energy–based P systems as a parallel and distributed model of computation in which the amount of energy manipulated and/or consumed during computations is taken into account. A given amount of energy is associated to each object of the system. Moreover, instances of a special symbol e are used to denote free energy units occurring into the regions of the system. These energy units can be used to transform objects, using appropriate rules. The rules are defined according to conservativeness considerations. An object can always be transformed into another object having the same energy. On the other hand, if the transformed object has a different energy then the required (resp., exceeding) free energy units are taken from (resp., released to) the region where the rule is applied. We assume that the application of rules consumes no energy. This means, in particular, that objects can be moved (without altering them) between the regions of the system without energy consumption. A special case of energy–based P systems are *conservative* P systems, where the amount of energy entering the system with the input values is completely returned with the output values at the end of the computation.

We first show that a Fredkin gate can be simulated using an energy–based P system. The proposed P system that performs the simulation turns out to be itself reversible and conservative. Subsequently, we show how any reversible circuit composed by Fredkin gates (also called *Fredkin circuits*, for short) can be simulated by a corresponding reversible and conservative energy–based P system. Indeed, the simulating P system can be made self–reversible, meaning that the same system can also perform backward computations. Since families of reversible Fredkin circuits can compute any function $f : \{0,1\}^* \to \{0,1\}$, energy–based P systems constitute the first known example (to the best knowledge of the authors) of reversible and universal P systems.

This is by no means the first time that energy is considered when dealing with P systems. We recall in particular [1,8,16,9]. This is not even the first paper which deals with the simulation of boolean gates and circuits by biologically inspired models of computation: for instance, in [13] a model for simulating boolean circuits (composed by AND, OR, NOT gates) with DNA algorithms is proposed, in [6] the same goal is reached using finite splicing, and in [5] some P systems that simulate boolean circuits are presented. Finally we also mention [10], where a biomolecular implementation of logically reversible computation using short strands of DNA as input and output lines of a Fredkin gate is demonstrated, and a method to connect Fredkin gates in order to create more complicated genetic networks is described.

2 Conservative Logic and Fredkin Circuits

Conservative logic is a mathematical model of computation based upon the so called *Fredkin gate*, a three–input/three–output boolean gate originally introduced by Petri in [17] whose input/output map FG : $\{0,1\}^3 \rightarrow \{0,1\}^3$ associates any input triple (x_1, x_2, x_3) with its corresponding output triple (y_1, y_2, y_3) as follows:

$$\begin{aligned} y_1 &= x_1 \\ y_2 &= (\neg x_1 \wedge x_2) \vee (x_1 \wedge x_3) \\ y_3 &= (x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3) \end{aligned} \tag{1}$$

A useful point of view is that the Fredkin gate behaves as a *conditional switch*: that is, $\mathrm{FG}(1, x_2, x_3) = (1, x_3, x_2)$ and $\mathrm{FG}(0, x_2, x_3) = (0, x_2, x_3)$ for every $x_2, x_3 \in \{0,1\}$. Hence, x_1 can be considered as a control input whose value determines whether the input values x_2 and x_3 have to be exchanged or not.

The Fredkin gate is *functionally complete* for boolean logic: by fixing $x_3 = 0$ we get $y_3 = x_1 \wedge x_2$, whereas by fixing $x_2 = 1$ and $x_3 = 0$ we get $y_2 = \neg x_1$.

The Fredkin gate is also *reversible*, that is, it computes a bijective map on $\{0,1\}^3$. Moreover, for every input/output pair the number of 1's in the input triple is the same as the number of 1's in the output triple. In other words, the output triple is obtained by applying an appropriate (input–dependent) permutation to the input triple. In [7] Fredkin and Toffoli interpret the conservation of the number of 1's between input and output triples as the conservation of the amount of energy associated to the input triple, thus assuming that two different triples having the same number of 0's and 1's require the same amount of energy to be realized in a physical system. Let us note that conservativeness is defined (both here and in [7]) as a mathematical notion; namely, it is *not* required that the entire energy used to perform the computation is preserved, or that the computing device is a conservative *physical* system (an ideal but unrealistic situation). In particular, we do not consider the energy needed to supply the computing device.

Putting together Fredkin gates we can build *Fredkin circuits*, that is, acyclic and connected directed graphs made up of *layers* of Fredkin gates. For a precise and formal definition of circuits see, for example, [19]. Figure 1 depicts an example of Fredkin circuit having three gates arranged in two layers. Evaluating

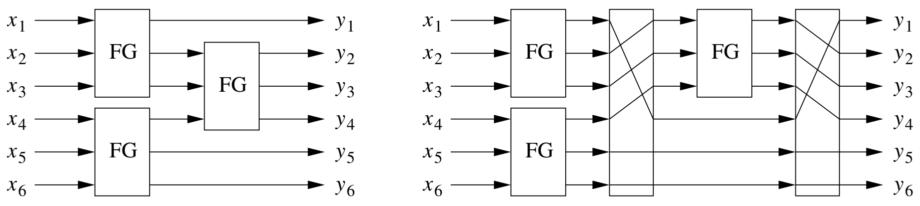


Fig. 1. A Fredkin circuit (on the left) and its normalized version

a Fredkin circuit in topological order (i.e. layer by layer, starting from the layer directly connected to the input lines) we can define the boolean function computed by the circuit as the composition of the functions computed by each layer of Fredkin gates. In evaluating the resources used by a Fredkin circuit to compute a boolean function we consider the *size* and the *depth* of the circuit, respectively defined as the number of gates and the number of layers of the circuit.

A *family* of Fredkin circuits is a sequence $\{FC_n\}_{n \in \mathbb{N}}$ where, for each $n \in \mathbb{N}$, FC_n is an n–input Fredkin circuit. Let f_{FC_n} denote the function computed by FC_n. Then we say that $\{FC_n\}_{n \in \mathbb{N}}$ computes the function $f : \{0,1\}^* \to \{0,1\}^*$ such that $f(\underline{x}) = f_{FC_{|\underline{x}|}}(\underline{x})$ for all $\underline{x} \in \{0,1\}^*$. Since the Fredkin gate is functionally complete, for any family $\{f_n\}_{n \in \mathbb{N}}$ of boolean functions there exists a family $\{FC_n\}_{n \in \mathbb{N}}$ of Fredkin circuits that computes it. Let $s, d : \mathbb{N} \to \mathbb{N}$. We say that family $\{FC_n\}_{n \in \mathbb{N}}$ has size s and depth d if for every $n \in \mathbb{N}$ the circuit FC_n has size $s(n)$ and depth $d(n)$.

A *reversible* n–input Fredkin circuit is a Fredkin circuit FC_n which computes a bijective map $f_{FC_n} : \{0,1\}^n \to \{0,1\}^n$. In a reversible Fredkin circuit the FanOut function, defined as $\text{FanOut}(x) = (x, x)$ for all $x \in \{0,1\}$, is explicitly computed with a gate. Fortunately, the Fredkin gate can also be used for this purpose, since $\text{FG}(x, 0, 1) = (x, x, \neg x)$ for $x \in \{0,1\}$. Compare this situation with usual (non reversible) circuits, where the FanOut function is simply implemented by splitting wires. Let us note that the function computed by a reversible Fredkin circuit is conservative.

In [7] Fredkin and Toffoli introduce a computational model for reversible and conservative computations. Computations are performed by reversible Fredkin circuits. The conservativeness requirement (preservation of the number of 1's) is again equivalent to the requirement that the output n-tuple is obtained by applying an appropriate (input–dependent) permutation to the corresponding input n-tuple.

It is important to note that reversibility and conservativeness are two independent notions: a function (computed by a gate or circuit) may be reversible, conservative, both or none of them. However, for any function $f : \{0,1\}^n \to \{0,1\}^m$ it is possible to build a new function $f_R : \{0,1\}^{n+m} \to \{0,1\}^{n+m}$ such that f_R is a bijection on $\{0,1\}^{n+m}$ and $f_R(\underline{x}, \underline{0}_m) = (\underline{x}, f(\underline{x}))$ for all $\underline{x} \in \{0,1\}^n$, where $\underline{0}_m$ is the m-tuple consisting of all 0's. Analogously, it is possible to build a conservative function f_C that computes the values assumed by f in its first m output bits. Finally it is also possible to extend the reversible function f_R built above to a reversible and conservative function f_{RC} by adding some additional input and output variables. For the proofs we refer the reader to [4].

3 Energy-Based P Systems

P systems (also called *membrane systems*) were introduced in [14] as a new class of distributed and parallel computing devices, inspired by the structure and functioning of living cells. The basic model consists of a hierarchical structure composed by several membranes, embedded into a main membrane called the *skin.* Membranes divide the Euclidean space into *regions*, that contain some *objects* (represented by symbols of an alphabet) and *evolution rules*. Using these rules, the objects may evolve and/or move from a region to a neighboring one. The rules are applied in a nondeterministic and maximally parallel way: all the objects that may evolve are forced to evolve. A *computation* starts from an initial configuration of the system and terminates when no evolution rule can be applied. The result of a computation is the multiset of objects contained into an *output membrane* or emitted from the skin of the system.

In what follows we assume the reader is already familiar with the basic notions and the terminology underlying P systems. For a systematic introduction, we refer the reader to [15]. The latest information about P systems can be found in [18].

In order to take into account the amount of energy used during computations, we define a new model which we call *energy–based P system.* In this model, we consider a special symbol e which denotes a free energy unit floating into regions; moreover, the rules are defined according to conservativeness considerations. We will show how this model can be used to simulate any reversible Fredkin circuit.

Formally, an energy–based P system (of degree $m \geq 1$) is a construct

$$\Pi = (A, \varepsilon, \mu, e, w_1, \ldots, w_m, R_1, \ldots, R_m, i_{\text{in}}, i_{\text{out}})$$

where:

- A is an alphabet; its elements are called *objects*;
- $\varepsilon : A \to \mathbb{N}$ is a mapping that associates to each object $a \in A$ the value $\varepsilon(a)$ (also denoted by ε_a), which can be viewed as the "energy value of a". If $\varepsilon(a) = \ell$, we also say that object a *embeds* ℓ units of energy;
- μ is a hierarchical membrane structure consisting of m membranes. For the sake of clarity, we will label membranes with mnemonic identifiers which recall their function;
- $e \notin A$ is a special symbol that denotes one *free energy* unit, that is, one unit of energy which is not embedded into any object;
- w_i, for all $i \in \{1, \ldots, m\}$, specify the multiset (over $A \cup \{e\}$) of objects initially present in region i;
- R_i, for all $i \in \{1, \ldots, m\}$, is a finite set of evolution rules over A associated with region i. Only rules of the following types are allowed:

  $$ae^k \to (b,p) \ , \qquad a \to (b,p)e^k \ , \qquad e \to (e,p) \ , \qquad a \to (b,p)$$

  where $a, b \in A$, $p \in \{\text{here}, \text{in}(name), \text{out}\}$ and k is a non negative integer;
- i_{in} is an integer between 1 and m and specifies the input membrane of Π;
- i_{out} is an integer between 0 and m and specifies the output membrane of Π. If $i_{\text{out}} = 0$ then the environment is used for the output, that is, the output value is the multiset of objects (over A) emitted from the skin.

A special attention is due to the definition of rules. The meaning of rule $ae^k \to (b,p)$, with $a, b \in A$, $p \in \{\text{here}, \text{in}(name), \text{out}\}$, and k a positive integer number, is the following: the object a, in presence of k free energy units, is allowed to be transformed into object b. If $p =$ here then the new object b remains in the same region; if $p =$ out then b exits from the current membrane. Finally, if $p = \text{in}(name)$ then b enters into the membrane labelled with *name*, which must be a child of the current membrane in the membrane hierarchy.

The meaning of rule $a \to (b,p)e^k$, when k is a positive integer number, is analogous. The object a is allowed to be transformed into object b by releasing k units of free energy. As above, the new object b may optionally move one level up or down into the membrane hierarchy. The k free energy units can now be used by another rule to produce "more energetic" objects from "less energetic" ones.

When $k = 0$ the rule $ae^k \to (b,p)$, also written as $a \to (b,p)$, transforms the object a into the object b and moves it (if $p \neq$ here) upward or downward into the membrane hierarchy, without acquiring nor releasing any free energy unit. Analogously, rules $e \to (e,p)$ simply move (if $p \neq$ here) one unit of free energy upward or downward into the membrane hierarchy.

A further constraint is that each rule must be "conservative", in the sense that the amount of energy occurring on the left side of the rule must be the same as the amount of energy which occurs on the right side.

With a little abuse of notation, when the pair (x,p), with $x \in A \cup \{e\}$ and $p \in \{\text{here}, \text{in}(name), \text{out}\}$, appears into a rule we will write x_p. Also, if $p = \text{in}(name)$ and no confusion arises we will usually write just the name of the membrane. Moreover, instead of writing e^k we will sometimes explicitly write k instances of e. It is also understood that the position of e^k (that is, on the left or on the right of the symbol from A) either into the left or into the right side of a rule is uninfluent. Finally, when the position p of an object which occurs in the right side of a rule is "here" we will omit to write it.

A *configuration* of Π is the sequence $(M_1, \ldots, M_m)$ of multisets (over $A \cup \{e\}$) of objects contained in each region of the system. $(w_1, \ldots, w_m)$ is called the *initial configuration.* For two configurations $(M_1, \ldots, M_m)$, $(M'_1, \ldots, M'_m)$ of Π we write $(M_1, \ldots, M_m) \Rightarrow (M'_1, \ldots, M'_m)$ to denote a *transition* from $(M_1, \ldots, M_m)$ to $(M'_1, \ldots, M'_m)$. The reflexive and transitive closure of $\Rightarrow$ is denoted by $\Rightarrow^*$. A *final configuration* is a configuration where no rule can be applied.

A *computation* is a sequence of transitions between configurations of Π, starting from the initial configuration. A computation is *successful* if and only if it reaches a final configuration or, in other words, it *halts.* It is understood that the multiset (over A, that is, not considering free energy units) of objects which occur in $w_{i_{\text{in}}}$ are the *input values* for the computation. Analogously, the multiset (over A) of objects occurring in the output membrane (or emitted from the skin if $i_{\text{out}} = 0$) in the final configuration is the *output* of the computation. A non–halting computation produces no output.

Since energy is an additive quantity, it is natural to define the *energy of a multiset* as the sum of the amounts of energy associated to each instance of the objects which occur into the multiset. Analogously, the energy of a configuration is the sum of the amounts of energy associated to each multiset which occurs into the configuration. A *conservative computation* is a computation where each configuration has the same amount of energy. A *conservative energy–based P system* is an energy–based P system that performs only conservative computations.

4 Simulating the Fredkin Gate with Energy-Based P Systems

As depicted in Figure 2, energy–based P systems can be used to simulate the Fredkin gate. The system contains 12 kinds of objects. For the sake of clarity,

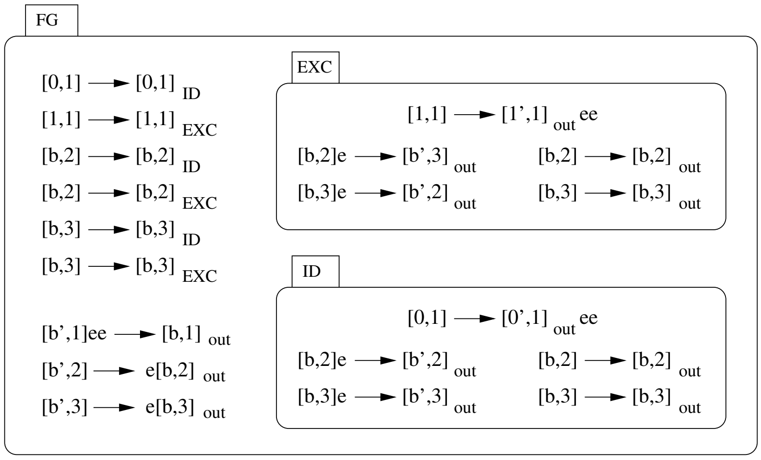


Fig. 2. Simulation of the Fredkin gate with an energy–based P system

we denote these objects by $[b, j]$ and $[b', j]$, with $b, b' \in \{0, 1\}$ and $j \in \{1, 2, 3\}$. Intuitively, $[b, j]$ and $[b', j]$ indicate the boolean value which occurs in the j-th line of the Fredkin gate. It will be clear from the simulation that we need two different symbols to represent each of these boolean values. Every object of the kind $[b, j]$, with $b \in \{0, 1\}$ and $j \in \{1, 2, 3\}$, has energy equal to 3, whereas the objects $[b', 1]$ have energy equal to 1 and the objects $[b', 2]$ and $[b', 3]$ (with $b' \in \{0, 1\}$) have energies equal to 4.

The simulation works as follows. The input values $[x_1, 1], [x_2, 2], [x_3, 3]$, with $x_1, x_2, x_3 \in \{0, 1\}$, are injected into the skin. If $x_1 = 0$ then the object $[0, 1]$ enters into membrane ID, where it is transformed to the object $[0', 1]$ by releasing 2 units of energy. The object $[0', 1]$ leaves membrane ID and waits for 2 energy units to transform back to $[0, 1]$ and leave the system. The objects $[x_2, 2]$ and $[x_3, 3]$, with $x_2, x_3 \in \{0, 1\}$, may enter nondeterministically either into membrane ID or into membrane EXC; however, if they enter into EXC they cannot be transformed to $[x'_2, 3]$ and $[x'_3, 2]$ since in EXC there are no free energy units. Thus the only possibility for objects $[x_2, 2]$ and $[x_3, 3]$ is to leave EXC and choose again between membranes ID and EXC in a nondeterministic way. Eventually, after some time they enter (one at the time or simultaneously) into membrane ID. Here they have the possibility to be transformed into $[x'_2, 2]$ and $[x'_3, 3]$ respectively, using the 2 units of free energy which occur into the region enclosed by ID (alternatively, they have the possibility to leave ID and choose nondeterministically between membranes ID and EXC once again). When the objects $[x'_2, 2]$ and $[x'_3, 3]$ are produced they immediately leave ID, and are only allowed to transform back to $[x_2, 2]$ and $[x_3, 3]$ respectively, releasing 2 units of energy. The objects $[x_2, 2]$ and $[x_3, 3]$ just produced leave the system, and the 2 units of energy can only be used to transform $[0', 1]$ back to $[0, 1]$ and expel it from the skin.

On the other hand, if $x_1 = 1$ then the object $[1, 1]$ enters into membrane EXC where it is transformed into the object $[1', 1]$ by releasing 2 units of energy. The object $[1', 1]$ leaves the membrane EXC and waits for 2 energy units to transform back to $[1, 1]$ and leave the system. Once again the objects $[x_2, 2]$ and $[x_3, 3]$, with $x_2, x_3 \in \{0, 1\}$, may choose nondeterministically to enter either into membrane ID or into membrane EXC. If they enter into ID they can only exit again since in ID there are no free energy units. When they enter into EXC they can be transformed to $[x'_2, 3]$ and $[x'_3, 2]$ respectively, using the 2 free energy units which occur into the region, and leave EXC. Now objects $[x'_2, 3]$ and $[x'_3, 2]$ can only be transformed into $[x_2, 3]$ and $[x_3, 2]$ respectively, and leave the system. During this transformation 2 free energy units are produced; these can only be used to transform $[1', 1]$ back to $[1, 1]$, which leaves the system.

It is apparent from the simulation that the system can be defined to work on any triple of lines of a circuit, simply modifying the values of the second component of the objects manipulated by the system.

The proposed P system is conservative: the number of energy units present into the system (both free and embedded into objects) during computations is constantly equal to 9. At the end of the computation, all these energy units are embedded into the output values. The system is also reversible: it is immediately seen that if we inject into the skin the output triple just produced as the result of a computation, the system will expel the corresponding input triple. This behavior is trivially due to the fact that the Fredkin gate is *self–reversible*, meaning that $\text{FG} \circ \text{FG} = \text{ID}_3$ (equivalently, $\text{FG} = \text{FG}^{-1}$), where ID_3 is the identity function on $\{0, 1\}^3$. Notice that, in general, this property does not hold for the functions $f : \{0, 1\}^n \to \{0, 1\}^n$ computed by n–input reversible Fredkin circuits. Indeed, f is self–reversible if and only if the permutation it applies on the set $\{0, 1\}^n$ can be expressed as a composition of pairwise disjoint transpositions. This means that in general the P system that simulates a given Fredkin circuit must be appropriately designed in order to be self–reversible.

5 Simulation of Reversible Fredkin Circuits

Basing upon the simulation of the Fredkin gate we have exposed in the previous section, let us sketch how any n–input reversible Fredkin circuit FC_n can be simulated by an appropriate energy–based P system P_n. Since families of reversible Fredkin circuits can be used to compute any family $\{f_n\}_{n\in\mathbb{N}}$ of n-ary boolean functions in a reversible and conservative way, we conclude that (families of) energy–based P systems are a *universal* model of computation.

Let $L_1, L_2, \ldots, L_d$ denote the layers of FC_n, where d is the depth of the circuit. The objects of P_n are denoted by $[b, i, j]$, where $b \in \{0, 1\}$, $i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, d+1\}$. All these objects have energy equal to 3. Intuitively, $[b, i, j]$ indicates the presence of the boolean value b on the i-th input line of the j-th layer of FC_n. The system P_n is composed by a main membrane (the skin) that contains a subsystem for each layer of FC_n. At the beginning of the computation objects $[x_1, 1, 1], [x_2, 2, 1], \ldots, [x_n, n, 1]$ are injected into the skin, where $(x_1, x_2, \ldots, x_n)$ is the input n-tuple of FC_n. The region associated to the skin contains the rules:

$$[b, i, j] \to [b, i, j]_{F_j} \qquad (2)$$

for every $b \in \{0, 1\}$, $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, d\}$, where F_j is the subsystem that simulates the j-th layer L_j. The application of these rules makes the objects representing the input values move into subsystem F_1.

Let k_1 be the number of gates in layer L_1. The subsystem F_1 contains k_1 subsystems $G_{1,1}, G_{1,2}, \ldots, G_{1,k_1}$, each one simulating a Fredkin gate as indicated in section 4. The only difference with the system presented in section 4 is that the objects now have an additional component indicating the number of layer, and when the results are expelled from subsystems $G_{1,1}, G_{1,2}, \ldots, G_{1,k_1}$ such component is incremented. Using rules of the kind $[b, i, 1] \to [b, i, 1]_{G_{1,r_i}}$ the objects are dispatched to the correct subsystems that perform the simulations of Fredkin gates. Eventually, after some time the objects corresponding to the result of the computation performed by each gate of L_1 leave the corresponding systems $G_{1,1}, G_{1,2}, \ldots, G_{1,k_1}$, with the third component incremented by 1. These objects are expelled from F_1 through appropriate rules present in the corresponding region. On the other hand, if a given object $[b, i, 1]$ hasn't to be processed by a Fredkin gate (that is, the identity must be applied to it) then we simply add the rule $[b, i, 1] \to [b, i, 2]_{out}$ to the region enclosed by membrane F_1. As objects $[b, i, 2]$ are expelled from F_1, rules (2) dispatch them to subsystem F_2.

The simulation of FC_n continues in this way until the objects $[b, i, d+1]$ leave the subsystem F_d. In the region enclosed by the skin they activate rules of the kind $[b, i, d+1] \to [b, i, d+1]_{out}$, that expel them into the environment as the result of the computation performed by P_n.

The system P_n is conservative, since the amount of energy units present into the system (both free and embedded into objects) during computations is constantly equal to $3n$. The number of rules and the number of membranes in the system are directly proportional to the number of gates in FC_n. Note that, differently from the other approaches seen in literature, the depth of hierarchy μ in system P_n is constant: in particular, it does not depend upon the number of gates occurring in FC_n.

5.1 Reverse Computations

Since the Fredkin circuit FC_n is reversible, there exists a Fredkin circuit FC'_n which computes the inverse function $f^{-1}_{FC_n} : \{0,1\}^n \to \{0,1\}^n$. This circuit can be easily obtained from FC_n by reversing the order of all layers. As a consequence, for any P system P_n which simulates an n–input reversible Fredkin circuit FC_n there exists a corresponding P system P'_n that simulates the inverse Fredkin circuit. In this sense, P_n can be considered a "reversible" P system.

However we can do slightly better, making the P system P_n self–reversible, that is, able to compute both f_{FC_n} and $f^{-1}_{FC_n}$. We add a further component $k \in \{0,1\}$ to the objects of P_n, which is used to distinguish between "forward" and "backward" computations. Precisely, the objects which are used to compute f_{FC_n} have $k = 0$, and those used to compute $f^{-1}_{FC_n}$ have $k = 1$. A forward computation starts by injecting the objects $[x_1, 1, 1, 0], [x_2, 2, 1, 0], \ldots, [x_n, n, 1, 0]$ into the skin of P_n. The computation proceeds as described above, with the rules modified in order to take into account the presence of the new component $k = 0$. The objects produced in output are $[y_1, 1, d + 1, 0], \ldots, [y_n, n, d + 1, 0]$, where $(y_1, \ldots, y_n) = f_{FC_n}(x_1, \ldots, x_n)$.

Analogously, a "backward" computation should start by injecting the objects $[y_1, 1, 1, 1], [y_2, 2, 1, 1], \ldots, [y_n, n, 1, 1]$ into the skin. The computation of $f^{-1}_{FC_n}$ can be accomplished by incorporating the rules of the region enclosed by the skin and the subsystems of P'_n (both modified in order to take into the account the presence of the new component $k = 1$) into P_n. Interferences between the rules concerning forward and backward computations do not occur since they act on different kinds of objects.

A further improvement is obtained by observing that each layer of FC_n is self–reversible, and that the layers of FC'_n are the same as the layers of FC_n, in reverse order. Hence we can merge each subsystem F_j, which simulates layer L_j of FC_n, with the subsystem F'_{d-j+1}, which simulates layer L'_{d-j+1} of FC'_n. The merge operation consists in putting the rules and the subsystems of F'_{d-j+1} into F_j. Of course we have also to modify the rules in the region enclosed by the skin so that the objects that were previously moved to F'_{d-j+1} are now dispatched to F_j. Recursively, since each Fredkin gate is self–reversible, we can merge also subsystems $G_{j,1}, \ldots, G_{j,k_j}$ occurring into F_j with the corresponding subsystems $G'_{d-j+1,1}, \ldots, G'_{d-j+1,k_j}$ which occur into F'_{d-j+1}. In this way, we obtain a self–reversible P system which is able to compute both f_{FC_n} and $f^{-1}_{FC_n}$. The new system has the same number of membranes as P_n, and the double of rules.

5.2 Reducing the Number of Subsystems

As we have seen in the previous sections, the number of membranes and the number of rules of the P system P_n that simulates the Fredkin circuit FC_n grow linearly with respect to the number of gates occurring in the circuit. This means, in particular, that if the size of the family $\{FC_n\}_{n \in \mathbb{N}}$ of Fredkin circuits grows exponentially with respect to n, also the number of membranes in the corresponding family $\{P_n\}_{n \in \mathbb{N}}$ of P systems will grow in an exponential way.

Here we note that the number of membranes in P_n can, without loss of generality, be assumed to be linear with respect to n, independently of the number of gates occurring in the simulated Fredkin circuit FC_n. To compensate the reduced number of membranes, the number of rules in the system will grow accordingly.

For the sake of simplicity, let us consider only forward computations, involving objects of the kind $[b, i, j]$, with $b \in \{0, 1\}$, $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, d+1\}$. Every n–input reversible Fredkin circuit FC_n can be "normalized" by moving the Fredkin gates contained into each layer as upward as possible, as depicted on the right of Figure 1. We call the resulting layers *normalized* layers. In order to keep track of which input value goes into which gate, we precede each normalized layer by a fixed permutation, which is simply realized by rearranging the wires as required. A final fixed permutation, occurring after the last normalized layer, allows the output values of FC_n to appear on the correct output lines.

It is easily seen that the described normalization of FC_n can be performed in polynomial time with respect to n. Also, it is not difficult to prove that the number of all possible n–input layers of Fredkin gates grows exponentially with n, whereas the number of normalized layers is $\lfloor \frac{n}{3} \rfloor$. We can thus number all possible normalized layers with an index $\ell \in \{1, \ldots, \lfloor \frac{n}{3} \rfloor\}$ and describe a normalized Fredkin circuit by a sequence of indexes $\ell_1, \ell_2, \ldots, \ell_d$ together with a corresponding sequence of fixed permutations $\pi_1, \pi_2 \ldots, \pi_{d+1}$.

The P system that simulates a normalized Fredkin circuit is thus composed by $\lfloor \frac{n}{3} \rfloor$ subsystems $F_1, \ldots, F_{\lfloor n/3 \rfloor}$, each one capable to simulate a fixed normalized layer of Fredkin gates. The region enclosed by the skin contains the rules $[b, i, j] \to [b, \pi_j(i), j]_{F_{\ell_j}}$ for all $b \in \{0, 1\}$, $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, d\}$, as well as the rules $[b, i, d+1] \to [b, \pi_{d+1}(i), d+1]_{out}$. These rules implement the fixed permutations, move the objects to the subsystem that simulates the next normalized layer, and expel the results of the computation into the environment. The simulation of each normalized layer is analogous to the simulation of the layers of a non normalized Fredkin circuit, as described above. Note that the objects emerge from subsystems $F_1, \ldots, F_{\lfloor n/3 \rfloor}$ with the j component incremented by 1, so that they are ready for the next computation step.

6 Conclusions

In this paper we have introduced energy–based P systems as P systems in which the amount of energy manipulated during computations is taken into account. We have also defined the notion of conservative energy–based P system.

We have shown how the Fredkin gate, as well as any n–input reversible Fredkin circuit can be simulated with this new model of computation. The P systems that perform these simulations turn out to be themselves reversible and conservative. Moreover, we have shown that the simulating P systems can be made self–reversible, and that the number of subsystems can be assumed to be linear with respect to the number of input values, independently of the number of gates occurring in the simulated Fredkin circuit.

Acknowledgments

The present paper has been inspired by [5] and by a discussion started by Fernando Sancho Caparrini during the Second Brainstorming Week on Membrane Computing, held in Seville from the 1st to the 7th of February 2004.

References

1. G. Alford. Membrane systems with heat control. In *Pre–Proceedings of the Workshop on Membrane Computing*, Curtea de Arges, Romania, August 2002.
2. J. D. Bekenstein. Energy Cost of Information Transfer. *Physical Review Letters*, 46(10):623–626, 1981.
3. C. H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17:525–532, November 1973.
4. G. Cattaneo, A. Leporati, R. Leporini. Fredkin Gates for Finite–valued Reversible and Conservative Logics. *Journal of Physics A: Mathematical and General*, 35:9755–9785, November 2002.
5. R. Ceterchi, D. Sburlan. Simulating Boolean Circuits with P Systems. In *Membrane Computing*, Proceedings of the International Workshop WMC 2003, Tarragona, Spain, July 2003, LNCS 2933, Springer, 2003, pp. 104–122.
6. K. Erk. Simulating Boolean Circuits by Finite Splicing. In *Proceedings of the Congress on Evolutionary Computation*, 2(6-9):1279–1285, IEEE Press, 1999.
7. E. Fredkin, T. Toffoli. Conservative Logic. *International Journal of Theoretical Physics*, 21(3-4):219–253, 1982.
8. R. Freund. Energy–Controlled P Systems. In *Membrane Computing*, Proceedings of the International Workshop WMC–CdeA 2002, Curtea de Arges, Romania, August 2002, LNCS 2597, Springer, 2002, pp. 247–260.
9. P. Frisco. The conformon–P system: a molecular and cell biology–inspired computability model. *Theoretical Computer Science*, 312:295–319, 2004.
10. J.P. Klein, T.H. Leete, H. Rubin. A biomolecular implementation of logically reversible computation with minimal energy dissipation. *Biosystems* 52:15–23, 1999.
11. R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5:183–191, 1961.
12. R. Landauer. Uncertainty principle and minimal energy dissipation in the computer. *International Journal of Theoretical Physics*, 21(3-4):283–297, 1982.
13. M. Ogihara, A. Ray. Simulating Boolean Circuits on a DNA Computer. *Tech. Report* 631, 1996. Available at: *http://citeseer.nj.nec.com/ogihara96simulating.html*
14. G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 1(61):108–143, 2000. See also Turku Centre for Computer Science – TUCS Report No. 208, 1998.
15. G. Păun. *Membrane Computing. An Introduction*. Springer–Verlag, Berlin, 2002.
16. G. Păun, Y. Suzuki, H. Tanaka. P Systems with energy accounting. *International Journal Computer Math.*, 78(3):343–364, 2001.
17. C. A. Petri. Gründsatzliches zur Beschreibung diskreter Prozesse. In Proceedings of the *3rd Colloquium über Automatentheorie (Hannover, 1965)*, Birkhäuser Verlag, Basel, 1967, pp. 121–140. English translation: Fundamentals of the Representation of Discrete Processes, ISF Report 82.04, 1982.
18. The P systems Web page: `http://psystems.disco.unimib.it/`
19. H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer–Verlag, 1999.

Solving 3CNF-SAT and HPP in Linear Time Using WWW

Florin Manea[1], Carlos Martín-Vide[2], and Victor Mitrana[1,2]

[1] Faculty of Mathematics and Computer Science, University of Bucharest
Str. Academiei 14, 70109, Bucharest, Romania
flmanea@funinf.cs.unibuc.ro

[2] Research Group in Mathematical Linguistics, Rovira i Virgili University
Pça. Imperial Tarraco 1, 43005, Tarragona, Spain
{cmv,vmi}@correu.urv.es

Abstract. We propose *linear* time solutions to two much celebrated NP-complete problems, namely the 3CNF-SAT and the directed Hamiltonian Path Problem (HPP), based on AHNEPs having all resources (size, number of rules and symbols) linearly bounded by the size of the given instance. Surprisingly enough, the time for solving HPP does not depend on the number of edges of the given graph. Finally, we discuss a possible real life implementation, not of biological inspiration as one may expect according to the roots of AHNEPs, but using the facilities of the World Wide Web.

1 Introduction

The origin of networks of evolutionary processors (NEPs for short) is twofold. In [7] we consider a computing model inspired by the evolution of cell populations, which might model some properties of evolving cell communities at the syntactical level. Cells are represented by words which encode their DNA sequences. Informally, at any moment of time, the evolutionary system is described by a collection of words, where each word represents one cell. Cells belong to species and their community evolves according to mutations and division which are defined by operations on words. Only those cells are accepted as surviving (correct) ones which are represented by a word in a given set of words, called the genotype space of the species. This feature parallels with the natural process of evolution.

On the other hand, a well-known architecture for parallel and distributed symbolic processing, related to the Connection Machine [12] as well as the Logic Flow paradigm [8], consists of several processors, each of them being placed in a node of a virtual complete graph, which are able to handle data associated with the respective node. Each node processor acts on the local data in accordance with some predefined rules, and then local data becomes a mobile agent which can navigate in the network following a given protocol. Only that data which can pass a filtering process can be communicated among the processors. This filtering process may require to satisfy some conditions imposed by the sending processor, by the receiving processor or by both of them. All the nodes send simultaneously their data and the receiving nodes handle also simultaneously all the arriving messages, according to some strategies, see, e.g., [9,12].

Starting from the premise that data can be given in the form of words, [5] introduces a concept called network of parallel language processors in the aim of investigating this concept in terms of formal grammars and languages. Networks of language processors are closely related to grammar systems, more specifically to parallel communicating grammar systems [4]. The main idea is that one can place a language generating device (grammar, Lindenmayer system, etc.) in any node of an underlying graph which rewrites the words existing in the node, then the words are communicated to the other nodes. Words can be successfully communicated if they pass some output and input filter. More recently, [6] introduces networks whose nodes are (standard) Watson-Crick D0L systems which communicate each other either the correct words or the corrected words.

In [1], we modify this concept in the following way inspired from cell biology. Each processor placed in a node is a very simple processor, an evolutionary processor. By an evolutionary processor we mean a processor which is able to perform very simple operations, namely point mutations in a DNA sequence (insertion, deletion or substitution of a pair of nucleotides). More generally, each node may be viewed as a cell having genetic information encoded in DNA sequences which may evolve by local evolutionary events, that is point mutations. Each node is specialized just for one of these evolutionary operations. Furthermore, the data in each node is organized in the form of multisets of words (each word appears in an arbitrarily large number of copies), and all copies are processed in parallel such that all the possible events that can take place do actually take place. Obviously, the computational process described here is not exactly an evolutionary process in the Darwinian sense. But the rewriting operations we have considered might be interpreted as mutations and the filtering process might be viewed as a selection process. Actually, many fitness functions on words may also be defined by random-context conditions. Recombination is missing but it was asserted that evolutionary and functional relationships between genes can be captured by taking only local mutations into consideration [15]. Consequently, hybrid networks of evolutionary processors might be viewed as bio-inspired computing models. We want to stress from the very beginning that we are not concerned here with a possible biological implementation, though a matter of great importance. However, in the last section we discuss an (im)possible and a bit funny implementation, not of biological inspiration as one may expected according to the above considerations, but using WWW.

Our mechanisms introduced in [1] are further considered in [2] as language generating devices and their computational power is investigated. Furthermore, filters, based on the membership and random-context conditions, used in [5] are generalized in some versions defined in [1,2,14]. More precisely, the new filters are based on different types of random-context conditions. In the aforementioned papers, the filters of all nodes are defined by the same random-context condition type. Moreover, the rules are applied in the same manner in all the nodes. These restrictions are discarded in [14] and [13]. By this reason, these networks were called *hybrid*.

In [13], we consider time complexity classes defined on accepting hybrid networks of evolutionary processors (AHNEP) similarly to the classical time complexity classes defined on the standard computing model of Turing machine. By definition, AHNEPs are deterministic. We prove that **NP** equals the class of languages accepted by AHNEPs in polynomial time.

In a series of papers, we present *linear* time solutions to some NP-complete problems using generating hybrid networks of evolutionary processors (GHNEP). Such solutions are presented for the Bounded Post Correspondence Problem in [1], for the "3-colorability problem" in [2] (with simplified networks), and for the Common Algorithmic Problem in [14].

This paper fits this line of research; we propose two linear time solutions to two much celebrated NP-complete problems, namely the 3CNF-SAT and the HPP, based on AHNEPs having all resources (size, number of rules and symbols) linearly bounded by the size of the given instance. However, this paper presents for the first time such solutions based on AHNEPs and not GHNEPs, and more important, by the definition of AHNEPs, one can evaluate the descriptional (number of nodes, rules, symbols) and computational (time) complexity of these AHNEPs with respect to their input word which is actually the given instance of the problem.

The paper is organized as follows. In the next section, we give the definition of basic preliminary concepts as well as AHNEP. Then, the third section presents a formal solution to 3CNF-SAT which runs in linear time on AHNEPs with linearly bounded resources. A similar solution to HPP is presented in the next section. These formal solutions are then "implemented" using the facilities of World Wide Web. A brief conclusion ends the paper.

2 Basic Definitions

We start by summarizing the notions used throughout the paper. An *alphabet* is a finite and nonempty set of symbols. The cardinality of a finite set A is written $card(A)$. Any sequence of symbols from an alphabet V is called *word* over V. The set of all words over V is denoted by V^* and the empty word is denoted by ε. The length of a word x is denoted by $|x|$ while $alph(x)$ denotes the minimal alphabet W such that $x \in W^*$.

We say that a rule $a \to b$, with $a, b \in V \cup \{\varepsilon\}$ is a *substitution rule* if both a and b are not ε; it is a *deletion rule* if $a \neq \varepsilon$ and $b = \varepsilon$; it is an *insertion rule* if $a = \varepsilon$ and $b \neq \varepsilon$. The set of all substitution, deletion, and insertion rules over an alphabet V are denoted by Sub_V, Del_V, and Ins_V, respectively.

Given a rule as above σ and a word $w \in V^*$, we define the following *actions* of σ on w:

- If $\sigma \equiv a \to b \in Sub_V$, then $\sigma^*(w) = \begin{cases} \{ubv : \exists u, v \in V^* \ (w = uav)\}, \\ \{w\}, \text{ otherwise} \end{cases}$
- If $\sigma \equiv a \to \varepsilon \in Del_V$, then $\sigma^*(w) = \begin{cases} \{uv : \exists u, v \in V^* \ (w = uav)\}, \\ \{w\}, \text{ otherwise} \end{cases}$

$$\sigma^r(w) = \begin{cases} \{u : \ w = ua\}, \\ \{w\}, \text{ otherwise} \end{cases} \qquad \sigma^l(w) = \begin{cases} \{v : \ w = av\}, \\ \{w\}, \text{ otherwise} \end{cases}$$

- If $\sigma \equiv \varepsilon \to a \in Ins_V$, then

$$\sigma^*(w) = \{uav : \ \exists u, v \in V^* \ (w = uv)\}, \ \sigma^r(w) = \{wa\}, \ \sigma^l(w) = \{aw\}.$$

$\alpha \in \{*, l, r\}$ expresses the way of applying a deletion or insertion rule to a word, namely at any position ($\alpha = *$), in the left ($\alpha = l$), or in the right ($\alpha = r$) end of the word, respectively. For every rule σ, action $\alpha \in \{*, l, r\}$, and $L \subseteq V^*$, we define the *α-action of σ on L* by $\sigma^\alpha(L) = \bigcup_{w \in L} \sigma^\alpha(w)$. Given a finite set of rules M, we define the *α-action of M* on the word w and the language L by:

$$M^\alpha(w) = \bigcup_{\sigma \in M} \sigma^\alpha(w) \ \text{ and } \ M^\alpha(L) = \bigcup_{w \in L} M^\alpha(w),$$

respectively. In what follows, we shall refer to the rewriting operations defined above as *evolutionary operations* since they may be viewed as linguistic formulations of local DNA mutations. For two disjoint and nonempty subsets P and F of an alphabet V and a word w over V, we define the predicates

$$\begin{aligned} &\varphi^{(1)}(w; P, F) \equiv P \subseteq alph(w) && \wedge \ F \cap alph(w) = \emptyset \\ &\varphi^{(2)}(w; P, F) \equiv alph(w) \subseteq P && \\ &\varphi^{(3)}(w; P, F) \equiv P \subseteq alph(w) && \wedge \ F \not\subseteq alph(w) \\ &\varphi^{(4)}(w; P, F) \equiv alph(w) \cap P \neq \emptyset && \wedge \ F \cap alph(w) = \emptyset. \end{aligned}$$

The construction of these predicates is based on *random-context conditions* defined by the two sets P (*permitting contexts/symbols*) and F (*forbidding contexts/symbols*). Informally, the first condition requires that all permitting symbols are and no forbidding symbol is present in w, the second one requires that all symbols of w are permitting ones, while the last two conditions are weaker variants of the first one such that some forbidding symbols may appear in w but not all of them, and at least one permitting symbol appears in w, respectively.

For every language $L \subseteq V^*$ and $\beta \in \{(1), (2), (3), (4)\}$, we define:

$$\varphi^\beta(L, P, F) = \{w \in L \mid \varphi^\beta(w; P, F)\}.$$

An *evolutionary processor over* V is a tuple (M, PI, FI, PO, FO), where:
– Either $(M \subseteq Sub_V)$ or $(M \subseteq Del_V)$ or $(M \subseteq Ins_V)$. The set M represents the set of evolutionary rules of the processor. As one can see, a processor is "specialized" in one evolutionary operation, only.
– $PI, FI \subseteq V$ are the *input* permitting/forbidding contexts of the processor, while $PO, FO \subseteq V$ are the *output* permitting/forbidding contexts of the processor.

We denote the set of evolutionary processors over V by EP_V. An *accepting hybrid network of evolutionary processors* (AHNEP for short) is a 7-tuple $\Gamma = (V, U, G, N, \alpha, \beta, x_I, x_O)$, where:

– V and U are the input and network alphabets, respectively, $V \subseteq U$.
– $G = (X_G, E_G)$ is an undirected graph with the set of vertices X_G and the set of edges E_G. G is called the *underlying graph* of the network.
– $N : X_G \longrightarrow EP_U$ is a mapping which associates with each node $x \in X_G$ the evolutionary processor $N(x) = (M_x, PI_x, FI_x, PO_x, FO_x)$.
– $\alpha : X_G \longrightarrow \{*, l, r\}$; $\alpha(x)$ gives the action mode of the rules of node x on the words existing in that node.
– $\beta : X_G \longrightarrow \{(1), (2), (3), (4)\}$ defines the type of the *input/output filters* of a node. More precisely, for every node, $x \in X_G$, the following filters are defined:

$$\begin{aligned}\text{input filter: } & \rho_x(\cdot) = \varphi^{\beta(x)}(\cdot; PI_x, FI_x),\\ \text{output filter: } & \tau_x(\cdot) = \varphi^{\beta(x)}(\cdot; PO_x, FO_x).\end{aligned}$$

That is, $\rho_x(w)$ (resp. τ_x) indicates whether or not the word w can pass the input (resp. output) filter of x. More generally, $\rho_x(L)$ (resp. $\tau_x(L)$) is the set of words of L that can pass the input (resp. output) filter of x.
– x_I and $x_O \in X_G$ is the *input node*, and the *output node*, respectively, of the AHNEP.

We say that $card(X_G)$ is the size of Γ. If $\alpha(x) = \alpha(y)$ and $\beta(x) = \beta(y)$ for any pair of nodes $x, y \in X_G$, then the network is said to be *homogeneous.* In the theory of networks some types of underlying graphs are common, e.g., *rings, stars, grids* etc. Networks of evolutionary processors with underlying graphs having these special forms have been considered in a series of papers [1,2,14,3]. We focus here on *complete* AHNEPs, i.e. AHNEPs having a complete underlying graph denoted by K_n, where n is the number of vertices.

A *configuration* of an AHNEP Γ as above is a mapping $C : X_G \longrightarrow 2^{V^*}$ which associates a set of words with every node of the graph. A configuration may be understood as the sets of words which are present in any node at a given moment. A configuration can change either by an *evolutionary step* or by a *communication step.* When changing by an evolutionary step, each component $C(x)$ of the configuration C is changed in accordance with the set of evolutionary rules M_x associated with the node x and the way of applying these rules $\alpha(x)$. Formally, we say that the configuration C' is obtained in *one evolutionary step* from the configuration C, written as $C \Longrightarrow C'$, iff

$$C'(x) = M_x^{\alpha(x)}(C(x)) \text{ for all } x \in X_G.$$

When changing by a communication step, each node processor $x \in X_G$ sends one copy of each word it has, which is able to pass the output filter of x, to all the node processors connected to x and receives all the words sent by any node processor connected with x providing that they can pass its input filter.

Formally, we say that the configuration C' is obtained in *one communication step* from configuration C, written as $C \vdash C'$, iff

$$C'(x) = (C(x) - \tau_x(C(x))) \cup \bigcup_{\{x,y\} \in E_G} (\tau_y(C(y)) \cap \rho_x(C(y))) \text{ for all } x \in X_G.$$

Note that words which leave a node are eliminated from that node. If they cannot pass the input filter of any node, they are lost.

Let Γ be an AHNEP, the computation of Γ on the input word $w \in V^*$ is a sequence of configurations $C_0^{(w)}, C_1^{(w)}, C_2^{(w)}, \ldots$, where $C_0^{(w)}$ is the initial configuration of Γ defined by $C_0^{(w)}(x_I) = w$ and $C_0^{(w)}(x) = \emptyset$ for all $x \in X_G$, $x \neq x_I$, $C_{2i}^{(w)} \Longrightarrow C_{2i+1}^{(w)}$ and $C_{2i+1}^{(w)} \vdash C_{2i+2}^{(w)}$, for all $i \geq 0$. Note that the two steps, evolutionary and communication, are synchronized and they happen alternatively one after another. By the previous definitions, each configuration $C_i^{(w)}$ is uniquely determined by the configuration $C_{i-1}^{(w)}$. Otherwise stated, each computation in an AHNEP is deterministic. A computation as above immediately halts if one of the following two conditions holds:

(i) There exists a configuration in which the set of words existing in the output node x_O is non-empty. In this case, the computation is said to be an *accepting computation.*
(ii) There exist two consecutive identical configurations.

In the aforementioned cases the computation is said to be finite. The language accepted by Γ is

$$L(\Gamma) = \{w \in V^* \mid \text{ the computation of } \Gamma \text{ on } w \text{ is an accepting one}\}.$$

We define some computational complexity measures by using AHNEP as the computing model. To this aim we consider a AHNEP Γ and the language L accepted by Γ. The *time complexity* of the accepting computation $C_0^{(x)}$, $C_1^{(x)}$, $C_2^{(x)}$, $\ldots C_m^{(x)}$ of Γ on $x \in L$ is denoted by $Time_\Gamma(x)$ and equals m. The time complexity of Γ is the partial function from $\mathbf{N}$ to $\mathbf{N}$,

$$Time_\Gamma(n) = \max\{Time_\Gamma(x) \mid x \in L(\Gamma), |x| = n\}.$$

For a function $f : \mathbf{N} \longrightarrow \mathbf{N}$ we define

$$\mathbf{Time}_{AHNEP}(f(n)) = \{L \mid L = L(\Gamma) \text{ for an AHNEP } \Gamma \text{ with } Time_\Gamma(n) \leq f(n) \text{ for some } n \geq n_0\}.$$

Moreover, we write $\mathbf{PTime}_{AHNEP} = \bigcup_{k \geq 0} \mathbf{Time}_{AHNEP}(n^k)$.

We recall the main result from [13]; the reader is referred to [10,11] for the classical time and space complexity classes defined on the standard computing model of Turing machine.

Theorem 1 [13] $\mathbf{NP} = \mathbf{PTime}_{AHNEP}$.

3 A Formal Solution to 3CNF-SAT in Linear Time Using AHNEPs

This section presents a formal way of getting all solutions to an instance of 3CNF-SAT using homogeneous AHNEPs. Satisfiability is perhaps the best studied NP-complete problem because one arrives at it from a large number of practical problems. It has direct applications in mathematical logic, artificial intelligence, VLSI engineering, computing theory, etc. It can also be met indirectly in the area of constraint satisfaction problems.

We are given a formula E in 3CNF with n variables and m clauses. The problem asks whether or not there exists an assignment of the n boolean variables such that the m clauses of 3 different variable each are all satisfied. In other words, the formula E is a conjunction (i.e., $\wedge$) of m clauses, with each being the disjunction (i.e., $\vee$) of three different variables or their negations (i.e., $\bar{\cdot}$) from a set of n variables. Let V be the set of variables, $V = \{x_1, x_2, \ldots, x_n\}$ and $E = C_1 \wedge C_2 \wedge \ldots C_m$ be a boolean formula in the 3CNF. The negation of a variable x_i is denoted by $\bar{x}_i$ and each clause may be viewed as a word over the alphabet $V \cup \bar{V} \cup \{\wedge, \vee, (,)\}$, where $\bar{V} = \{\bar{x} \mid x \in V\}$. We define the alphabet

$$W = V \cup \{1\} \cup \{[x_i = 1], [x_i = 0] \mid 1 \le i \le n\} \cup \{(s(C_i)) \mid 1 \le i \le m\},$$
where s is a finite substitution defined by $s(y) = \{0, y\}, y \in V \cup \bar{V}$,
$s(\circ) = \{\circ\}, \circ \in \{\wedge, \vee\}$.

We now consider the AHNEP

$$\Gamma = (V \cup \{(C_i) \mid 1 \le i \le m\}, W, K_{2n+2}, N, \alpha, \beta, In, Out),$$

where K_{2n+2} is the complete graph with $2n + 2$ nodes and the other parameters are given in Table 1.

Node	M	PI	FI	PO	FO	α	β
In	$\emptyset$	$\emptyset$	W	$\emptyset$	$\emptyset$	$*$	(1)
$N(x_i = 1)$	$P(x_i = 1)$	$\emptyset$	$\{[x_i = 1], [x_i = 0]\}$	$\emptyset$	$\{x_i\} \cup \{(\alpha) \mid x_i \prec \alpha\}$	$*$	(1)
$N(x_i = 0)$	$P(x_i = 0)$	$\emptyset$	$\{[x_i = 1], [x_i = 0]\}$	$\emptyset$	$\{x_i\} \cup \{(\alpha) \mid x_i \prec \alpha\}$	$*$	(1)
Out	$\emptyset$	$\emptyset$	$F(Out)$	$\emptyset$	W	$*$	(1)

Table 1.

In this table, the following hold:

(i) $1 \le i \le n$.
(ii) (α) is a generic symbol in the alphabet $\{(s(C_j)) \mid 1 \le j \le m\}$ with the substitution s from above.
(iii) $P(x_i = 1) = \{x_i \to [x_i = 1]\} \cup \{(\beta) \to \begin{cases} (g_i(\beta)), & \text{if } \bar{x}_i \text{ occurs in } \beta \\ 1, & \text{if } x_i \text{ occurs in } \beta \end{cases}$, where g_i is a morphism defined by $g_i(\bar{x}_i) = 0$ and $g_i(z) = z$ for all $z \in (V \cup \bar{V} \cup \{\wedge, \vee, (,)\}) \setminus \{x_i, \bar{x}_i\}$.

(iv) $P(x_i = 0) = \{x_i \to [x_i = 0]\} \cup \{(\alpha) \to \begin{cases} (\bar{g}_i(\beta)), & \text{if } x_i \text{ occurs in } \beta \\ 1, & \text{if } \bar{x}_i \text{ occurs in } \beta \end{cases}$, where $\bar{g}_i$ is a morphism defined by $\bar{g}_i(x_i) = 0$ and $\bar{g}_i(z) = z$ for all $z \in (V \cup \bar{V} \cup \{\wedge, \vee, (,)\}) \setminus \{x_i, \bar{x}_i\}$.

(v) $F(Out) = \{(s(C_j)) \mid 1 \le j \le m,\ s \text{ is defined as above}\} \cup \{(0 \vee 0 \vee 0)\}$.

(vi) The relation $x_i \prec \alpha$ means that the symbol x_i appears in the word α.

Let us outline the working mode of Γ on the input word

$$w = x_1 x_2 \dots x_n (C_1)(C_2) \dots (C_m).$$

For a better understanding we refer to the word $x_1 x_2 \dots x_n$ as the n-prefix of w, while $(C_1)(C_2)\dots(C_m)$ is called the m-suffix of w. In the initial configuration this word lies in the input node In. After an evolutionary step without any effect on this word, a copy of w is sent to all the nodes excepting In. By the input filter conditions, each node $N(x_i = 1)$ and $N(x_i = 0)$, $1 \le i \le n$, receives a copy of w, while the copies of w sent to the other nodes fail to pass their input filter condition, hence they are lost. Let us follow a copy of w arrived in the node $N(x_i = 1)$ for some $1 \le i \le n$. This word remains here until the following conditions are satisfied:

1. x_i is replaced by $[x_i = 1]$ in the n-prefix of w.
2. Every occurrence of $\bar{x}_i$ in any word C_j, $1 \le j \le m$, is replaced by 0.
3. Every symbol in the m-suffix of w which contains x_i is replaced by 1.

This phase takes $2r_i + 1$ steps ($r_i + 1$ evolutionary steps and r_i communication ones), where r_i is the number of occurrences of both x_i and $\bar{x}_i$ in the m-suffix of w. Now a copy of

$$w' = x_1 x_2 \dots x_{i-1}[x_i = 1]x_{i+1} \dots x_n (C'_1)(C'_2) \dots (C'_m)$$

is sent to every node. Here (C'_j) is either 1, if C_j contains x_i, or $(g_i(C_j))$, if $\bar{x}_i$ occurs in C_j, or (C_j), otherwise. Clearly, every node $N(x_k = 1)$ and $N(x_k = 0)$, $1 \le k \ne i \le n$, receives this word. After at most $2r_k + 1$ steps, where r_k is the number of occurrences of both x_k and $\bar{x}_k$ in the m-suffix of w (which is at least the number of occurrences of both x_k and $\bar{x}_k$ in the m-suffix of w') the resulting word which has either $[x_k = 1]$ or $[x_k = 0]$ instead of x_k and no occurrence of x_k and $\bar{x}_k$ in any symbol from its m-suffix, is replicated and one copy is sent again to all the nodes. This process lasts for at most $2n + 2\sum_{j=1}^{n} r_j = 2n + 6m$ steps. Then every word which does not contain any symbol from V in its n-prefix and has the m-suffix 1^m is simultaneously received by the output node Out and the computation halts. Moreover, any other word is rejected by Out. It is plain that if there are assignments which satisfy the formula E, the computation of Γ halts as soon as Out receives at least one word having an n-prefix which gives such an assignment. Such a computation lasts for at most $2n + 6m + 1$ steps. If there is no assignment satisfying E, then the computation of Γ halts after $2n + 6m$ steps since $C^{(w)}_{2n+6m}(y) = C^{(w)}_{2n+6m+1}(y)$ for any node y.

We want to stress that also the other resources of Γ are linearly bounded: the number of symbols in W is $3n + 8m + 1$ while the total number of rules is $2n + 6m$. This follows immediately from the fact that each set $P(x_i = 1)$ and $P(x_i = 0)$, $1 \le i \le n$, has $r_i + 1$ rules. It is worth mentioning the fact that the underlying structure does not change if the number of variables in the given instance remains the same. We also can say that the network, excepting the input and output nodes, may be viewed as a "program". In other words, the underlying structure of Γ is common for any instance of 3CNF-SAT with a fixed number of variables.

4 A Formal Solution to HPP in Linear Time Using AHNEPs

The HPP is to decide whether or not a given directed graph has a Hamiltonian path. A Hamiltonian path in a directed graph is a path which contains all vertices exactly once. It is known that the HPP is an NP-complete problem. Let us consider a directed graph $G = (V, E)$, with $V = \{x_1, x_2, \ldots, x_n\}$ for which we are looking for a Hamiltonian path starting with x_1. Assume that all edges going out from the node x_j, for some $1 \le j \le n$, are

$$(x_j, x_{i_1}^{(j)}), (x_j, x_{i_2}^{(j)}), \ldots, (x_j, x_{i_{k_i}}^{(j)}) \in E.$$

First we define the alphabet

$$U = V \cup E \cup \{[x_i] \mid 1 \le i \le n\} \cup \{[x, y] \mid (x, y) \in E\}$$

and the the AHNEP

$$\Gamma = (V \cup E, U, K_{2n+1}, \mathcal{N}, \alpha, \beta, In, Out),$$

where K_{2n+1} is the complete graph with $2n + 1$ nodes and the other parameters are given in Table 2.

Node	M	PI	FI	PO	FO	α	β
In	$\{x_1 \to [x_1]\}$	$\emptyset$	U	$\emptyset$	$\emptyset$	$*$	(1)
$N(x_i)$	$\{x_i \to [x_i]\}$	$\{[x_j, x_p^{(j)}] \mid 1 \le j \le n, x_p^{(j)} = x_i\}$	$\{[x_i]\}$	U	$\emptyset$	$*$	(4)
$N'(x_j)$	$\{(x_j, x_{i_t}^{(j)}) \to [x_j, x_{i_t}^{(j)}] \mid 1 \le t \le k_j\}$	$\{[x_j]\}$	$\{[x_j, x_{i_t}^{(j)}] \mid 1 \le t \le k_j\}$	$\emptyset$	$\emptyset$	$*$	(1)
Out	$\emptyset$	$\emptyset$	V	$\emptyset$	U	$*$	(1)

Table 2.

Let us follow a computation of this AHNEP on the input word

$$x_1 x_2 \ldots x_n (x_1, x_{i_1}^{(1)})(x_1, x_{i_2}^{(1)}) \ldots (x_1, x_{i_{k_1}}^{(1)}) \ldots (x_n, x_{i_1}^{(n)})(x_n, x_{i_2}^{(n)}) \ldots (x_n, x_{i_{k_n}}^{(n)}).$$

The input word is referred as w. In the node In, x_1 is replaced by $[x_1]$, the new word w' is sent out and $N'(x_1)$ is the unique node that can receive it. Here each symbol $(x_1, x_{i_t}^{(1)})$, with $1 \leq t \leq k_1$, is replaced in different copies of w' by $[x_1, x_{i_t}^{(1)}]$. Now, each node $N(x_j)$ such that $x_j = x_{i_t}^{(1)}$ receives exactly one of the words going out from $N'(x_1)$. Note that the number of words going out from $N'(x_1)$ is exactly k_1. In every node $N(x_j)$, x_j is replaced by $[x_j]$ and the aforementioned process resumes. Note that for every node there exist exactly two consecutive configurations in which that node has a nonempty set of words. By these informal explanations, we infer that if $\pi = x_1 x_{j_1} \dots x_{j_r}$ is a path in G, then

$$h(x_1 x_2 \dots x_n)\alpha \in C_{4r+1}^{(w)}(N(x_{j_r})),$$

where h is a morphism that replaces the symbols $x_1, x_{j_1}, \dots, x_{j_r}$ by $[x_1], [x_{j_1}], \dots, [x_{j_r}]$, respectively, and leaves unchanged the other symbols, while $\alpha \in (E \cup \{[x,y] \mid (x,y) \in E\})^+$. Furthermore, α contains exactly r symbols from $\{[x,y] \mid (x,y) \in E\}$, namely the copies of the symbols representing the edges of the path π. Conversely, one can prove by induction on r that whenever such a word enters a node $N'(x)$ or goes out from a node $N(x)$, $x_1 x_{j_{\sigma(1)}} \dots x_{j_{\sigma(r)}}$ is a path in G for some permutation σ of the first r naturals. Since any word $[x_1][x_2]\dots[x_n]\beta$ is accepted by Out, the correctness of our construction follows. More precisely, there exists a configuration in which Out has at least one word if and only if G has a Hamiltonian path.

Note that Γ halts after at most $4n$ steps, therefore Γ provides an answer in $O(n)$ time. This is quite surprising since the time needed by Γ for checking the existence of a Hamiltonian path in G does not depend on the number of edges of G. Finally, the number of symbols and rules of Γ is $2n + 2m$ and $n + m$, respectively.

5 A(n) (Im)Possible Implementation of the Formal Solutions Using E-mail

We refer to the solution for the 3CNF-SAT, a similar implementation for the solution to HPP is obvious. We place in any node of the above network a person who is writing e-mail messages. All the persons in the networks know each other (they are friends) and each of them has the e-mail address of all the others stored in his/her address book. For the given instance of 3CNF-SAT from above, the network is formed by $2n+2$ persons: let us call them by In, Out, $True_1$, $True_2$, $\dots$, $True_n$, and $False_1$, $False_2$, $\dots$, $False_n$. Each person $True_i$ and $False_i$ has configured his/her SPAM filter such that incoming messages are rejected if they do not satisfy the random-context conditions determined by the forbidding symbols in the sets FI from Table 1. This is an easy task since it requires just to check the absence of some symbols from a finite set in the incoming messages.

By a bit more complicated procedure, each person can also configure his/her e-mail software such that an outgoing message cannot be sent unless it satisfies the random-context conditions defined by the sets FO in Table 1. Now the working protocol is the following. Initially, In sends w to all his/her friends. Every receiver waits for a predefined period of time until the messages sent by his/her friends have been collected in his/her incoming box. This period is settled such that it suffices for all persons. Then each person works in parallel on his/her new incoming messages. Each message is modified in accordance with the substitution rules defined by the sets M in Table 1. This phase lasts again for a predefined period of time sufficiently long such that all persons can finish their task within this period. Note that the synchronization is an important part of the computation, however this phase can also be carried out by a not very complicated software. Then, each of them sends simultaneously the modified messages to the others and the process resumes. After a while this process halts since Out, who has been counting the steps, sends simultaneously a message asking for stopping the process to all the others as soon as either its incoming box contains at least one message or the number of steps exceeded $2n + 6m$. If the incoming box of Out is non-empty, it contains at least one solution to the given instance of 3CNF-SAT. The messages that were successfully sent are stored in the outgoing box so that they do not appear in the incoming box anymore. A small problem is represented by the messages that were sent but rejected by some recipients which might come back in the incoming box. It is easy to configure the e-mail software such that as soon as a message was sent it cannot come back.

A serious problem is represented by the time needed for modifying the new messages as well as the possibility of storing these messages since the number of new messages received by each person is doubled in any step.

6 Conclusion

We presented formal solutions to two NP-complete problem, 3CNF-SAT and HPP, running in linear time on AHNEPs. Then we discussed an (im)possible implementation using the facilities of WWW. Note that the constructed AHNEPs are rather simple which proves once more that the computational power of these devices is high. Actually, the problem of finding the class of all NP problems that can be solved in linear time by AHNEPs with linearly bounded resources seems to be quite attractive. We hope to return to this topic in a forthcoming work.

References

1. J. Castellanos, C. Martin-Vide, V. Mitrana, J. Sempere, Solving NP-complete problems with networks of evolutionary processors, *IWANN 2001* (J. Mira, A. Prieto, eds.), LNCS 2084, Springer-Verlag, 2001, 621–628.
2. J. Castellanos, C. Martin-Vide, V. Mitrana, J. Sempere, Networks of evolutionary processors, *Acta Informatica* 39(2003), 517-529..
3. J. Castellanos, P. Leupold, V. Mitrana, Descriptional and computational complexity aspects of hybrid networks of evolutionary processors, *Theoretical Computer Science*, in press.

4. E. Csuhaj-Varjú, J. Dassow, J. Kelemen, G. Păun, *Grammar Systems*, Gordon and Breach, 1993.
5. E. Csuhaj-Varjú, A. Salomaa, Networks of parallel language processors. In: *New Trends in Formal Languages* (G. Păun, A. Salomaa, eds.), LNCS 1218, Springer Verlag, 1997, 299–318.
6. E. Csuhaj-Varjú, A. Salomaa, Networks of Watson-Crick D0L systems. In: *Proc. International Conference Words, Languages & Combinatorics III* (M. Ito, T. Imaoka, eds.), World Scientific, Singapore, 2003, 134–150.
7. E. Csuhaj-Varjú, V. Mitrana, Evolutionary systems: a language generating device inspired by evolving communities of cells, *Acta Informatica* 36(2000), 913–926.
8. L. Errico, C. Jesshope, Towards a new architecture for symbolic processing. In *Artificial Intelligence and Information-Control Systems of Robots '94* (I. Plander, ed.), World Sci. Publ., Singapore, 1994, 31–40.
9. S. E. Fahlman, G. E. Hinton, T. J. Seijnowski, Massively parallel architectures for AI: NETL, THISTLE and Boltzmann machines. In *Proc. AAAI National Conf. on AI*, William Kaufman, Los Altos, 1983, 109–113.
10. J. Hartmanis, P.M. Lewis II, R.E. Stearns, Hierarchies of memory limited computations. *Proc. 6th Annual IEEE Symp. on Switching Circuit Theory and Logical Design*, 1965, 179 - 190.
11. J. Hartmanis, R.E. Stearns, On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* 117 (1965), 533–546.
12. W. D. Hillis, *The Connection Machine*, MIT Press, Cambridge, 1985.
13. M. Margenstern, V. Mitrana, M. Perez-Jimenez, Accepting hybrid networks of evolutionary processors, *Pre-proccedings of DNA 10*, 2004, 107–117.
14. C. Martin-Vide, V. Mitrana, M. Perez-Jimenez, F. Sancho-Caparrini, Hybrid networks of evolutionary processors, *Proc. of GECCO 2003*, LNCS 2723, Springer Verlag, Berlin, 401 - 412..
15. D. Sankoff et al. Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. *Proc. Natl. Acad. Sci. USA*, 89(1992) 6575–6579.

Completing a Code in a Regular Submonoid of the Free Monoid

(Extended Abstract)

Jean Néraud

LIFAR, Université de ROUEN, Faculté des Sciences, Place É. Blondel, F-76821
MONT-SAINT-AIGNAN-CEDEX, FRANCE

Abstract. Let M be a submonoid of the free monoid A^*, and let $X \subset M$ be a (variable length) code. X is weakly M-complete iff any word in M is a factor of some word in X^* [NS 03]. In this paper, we are interested by an effective computation of a weakly M-complete code containing X. Given a regular submonoid and given a code $X \subset M$, we present a method of completion which preserves the regularity of X.

Keywords: Words, free monoid, submonoid, code, automaton, completeness, density

1 Introduction

In the free monoid theory, the questions connected to maximality and completeness of variable length codes (for short codes) play a prominent part. This is due to their mathematical relevance just as much as their potential applications, particularly in the framework of the theory of Information. A subset X of the free monoid A^* is complete if any word of A^* is a factor of some word of X^*, the submonoid generated by X. In this topic, one of the most notable results, due to Schützenberger, states that, for the remarkable large family of thin codes [BerP 85, p. 65], completeness and maximality are two equivalent notions. Based on this fact, many fruiful studies have been drawn in various directions. In a large lot of these studies authors have investigated whether the preceding equivalence holds for special families of codes, e.g. [BerP 85, BrWZ 90, Br 98, Nh 01, NS 03]. The question of investigating local notions of completeness is also largely concerned:

- In the topic of codes with constraints, that is strongly connected to sofic systems [Bea 93], A. Restivo studies the class of the so-called local languages. These languages are in fact the factorial subsets of the form $T = A^* \setminus A^*HA^*$, where H stands for a finite language. In such a context, given a code $X \subset T$, X is T-complete iff $F(X^*)$, the set of the factors of the words in X^*, is equal to $F(T)$ itself. As a matter of fact, in [R 90], the equivalence between T-completeness and maximality in T is established for the so-called T-thin codes.

- The topic of infinitary languages is also concerned by another notion of local completeness: given a language $R \subset A^*$, the problem consists in studying the structure of languages $G \subset A^*$ such that $G^\omega = R^\omega$ or, more generally,

${}^{\omega}G^{\omega} = {}^{\omega}R^{\omega}$, which is in fact equivalent to G^* and R^* having identical sets of prefixes [LaT 86, LiT 87, Li 91] or factors [DevL 91]. Note that the elements of the generating set G are not required to be words of R^*.

- Our paper deals with local completeness in an arbitrary submonoid M of A^*: given a set $X \subset M$, X is weakly M-complete iff M and X^* have identical sets of factors. This notion is in fact more general than the notion of completeness in a monoid as defined in [BerP 85]. Its introduction is justified by the fact that, as explained in [R 89, R 90], given the language $L(S)$ of the finite blocs of a sofic system S, studying $L(S)$-completeness comes down to study completeness in particular submonoids of $L(S)$, i.e. the so-called stabilizers. Moreover, as illustrated in [NS 03], many properties of codes in A^*, beginning by the famous theorem of Shützenberger, may be established as consequences of more general results from the framework of arbitrary submonoids. In this context, a natural question consists in examining the existence of weakly M-complete codes, given a submonoid M of A^*. In [BlH 85], a positive answer is given. In the case where M is a regular submonoid, the existence of a prefix circular $L(S)$-complete code is established in [Bea 93], moreover, according to [LaT 86, LiT 87, Li 91, NS 03A], the existence of prefix (weakly) M-complete codes is decidable. However, the existence of a regular weakly M-complete code itself remained an open question [NS 03].

According to Zorn's lemma, given an arbitrary submonoid M, and given a code $X \subset M$, the family of codes X' that satisfy $X \subset X' \subset M$ has a maximal element. This proprety, conjugated with a result in [NS 03] allows to establish (in a non constructive way) the existence, for any code $X \subset M$, of a weakly M-complete code, namely $\hat{X}$, such that $X \subset \hat{X} \subset M$.
In our paper we consider the case where the submonoid has "finite rank". This notion of rank is in fact defined with respect to the minimal automaton with behavior M (which constitutes a canonical representation of M). It corresponds to the smallest positive integer r such that a word $w \in A^*$ which satisfies $|Q.w| = r$ exists (where Q stands for the set of states). It is important to note that, with this definition, states $q \in Q$ such that $q.w$ is not defined may exist. As a remarkable example, any regular submonoid of A^* has finite rank.
Given a submonoid with finite rank $M \subset A^*$, and given a code $X \subset M$, we present a constructive method for embedding X in a weakly M-complete code, namely $\hat{X}$. As a direct consequence, since our computation preserves the regularity of sets, given a regular submonoid M of A^*, a general formula for computing a regular M-complete code is guaranteed: this brings an affirmative answer to the question that we formulated in [NS 03].

2 Preliminaries

2.1 Definitions and Notations

We adopt the standard notations of the free monoid theory : given a word w in A^* (the free monoid generated by A), we denote by $|w|$ its length, the empty word being the word of length zero.

Given two words $u, w \in A^*$, we say that u is a *factor (internal factor, prefix, proper prefix, suffix)* of w iff we have $w \in A^*uA^*$ (A^+uA^+, uA^*, uA^+, A^*u). Given a subset X of A^*, we denote by $F(X)$ ($P(X), S(X)$), the set of the words that are factor (prefix, suffix) of some word in X.
Given a word $w \in A^+$, its primitive root is the shortest word x such that $w \in x^+$. If $w = x$ we say that w is *primitive*, otherwise it is *imprimitive*. The following characterization of primitiveness is of folklore:

$$w \in A^* \text{ is primitive } \iff w \text{ is not an internal factor of } w^2 \quad (1)$$

Let $U, V \subset A^*$, and let $w \in A^*$. We say that w is $U \times V$*-overlapping* if a pair of words $u \in U$, $v \in V$ exists, such that the three conditions $u \neq v$, $vw \in P(uw)$ and $u \in P(vw)$ hold. Otherwise, we say that w is $U \times V$*-overlapping free.* Note that the word w may be $U \times V$-overlapping, unless it is $V \times U$-overlapping. Clearly, in the case where we have $U = V = A^*$, this notion of $U \times V$-overlapping comes down to the classical notion of overlapping.

2.2 Density in a Submonoid of A^*

We assume the reader familiar with the fundamental concepts of the theory of variable length codes, and we suggest that he refers to [BerP 85]. Let M be a submonoid of A^*. As commented in [NS 03], different types of densities may be defined in M. In our paper, we are interested by weak density. Note that this notion is in fact more general that the algebraical concept of density as presented in [BerP 85], in which a set $X \subset M$, is M-dense (M-complete) iff for any word $w \in M$, we have $MwM \cap X \neq \emptyset$ ($MwM \cap X^* \neq \emptyset$).

Definition 1. *Let M be an arbitrary submonoid of A^*, and let $X \subset M$.*
(1) X is weakly dense in M if for any word $w \in M$ we have $A^*wA^* \cap X \neq \emptyset$.
(2) X is strongly M-thin if a word $w \in M$ exists such that $A^*wA^* \cap X = \emptyset$.
(3) X is weakly M-complete if X^* is weakly dense in M.
(4) X is a M-maximal code iff it is a code, and for any code $X' \subset M$, if X' contains X, then we have $X' = X$.

As shown by the following result, for strongly M-thin sets the two notions of weakly M-complete set and M-complete set are in fact equivalent:

Theorem 1. [NS 02] *Let $X \subset M$ a strongly M-thin set. Then X is weakly M-complete iff for any word $y \in M$, we have*

$$(X^*y)^+X^* \cap X^* \neq \emptyset.$$

One of the most important results that illustrate the theory of codes is the theorem of Schützenberger, that has been generalized to several classes of codes. In particular:

Theorem 2. [NS 03] *Let $M \subset A^*$ be an arbitrary submonoid of A^* and let $X \subset M$ be a strongly M-thin code. Then the three following properties are equivalent:*
(i) X is weakly M-complete
(ii) X is M-complete
(iii) X is a M-maximal code.

The proof of Theorem 2, lays upon Theorem 1, and the following technical lemmas, that are established in [NS 03]. These results will be of peculiar importance in the sequel of the paper.

Lemma 1. [NS 03] *Let $X \subset M$. If X is not weakly M-complete then for any word $t \in M \setminus F(X^*)$, a word $z \in M$ exists such that the two following conditions hold:*
(i) z and t have different primitive roots
(ii) For any integer $k > \frac{|t|}{|z|} + 2$, $y = z^k t$ is a primitive word.

Lemma 2. *Let $X \subset M$ and let $y \in M \setminus F(X^*)$ a primitive word. Then the word $z = y^2$ is a $X^* \times A^*$-overlapping free word in $M \setminus F(X^*)$.*

Lemma 3. *Any M-maximal code is weakly M-complete.*

3 A Property of Submonoids with Finite Rank

3.1 The Case Where M Is a Free Submonoid of A^*

In such a condition, a coding isomorphism ϕ, from a free monoid B^* onto M, exists. By applying the classical construction from [ER 83], a B^*-complete code Y containing $\phi^{-1}(X)$ exists. Recall that the computation makes use of the existence of an overlapping free word $y \notin F(X^*)$ [BerP 85, p.10]. It consists in embedding $Y = \phi^{-1}(X)$ in the complete code $\hat{Y} = Y \cup y(Vy)^*$, with $V = B^* \setminus Y^* \setminus B^*yB^*$. This leads to embed X itself in the (weakly) M-complete code $\hat{X} = \phi(\hat{Y})$.

Example 1. Let $A = \{a, b\}$ and let $M = \{a^n b^n | n \geq 1\}^*$. M is a free (bi-unitary) submonoid of A^*, hence we may apply the preceding construction. Given the code $X = \{aba^2b^2\}$, the computation yields to $\hat{X} = X \cup a^3b^3[(M \setminus X^* \setminus Ma^3b^3M)a^3b^3]^*$.

3.2 Words with Minimal Rank in a Submonoid of A^*

Let M be a free submonoid of A^* and let $X \subset M$ be a code. According to Zorn's lemma, a M-maximal thus, according to Lemma 3, a weakly M-complete code containing X exists. However, since the existence of a $M \times M$-overlapping free word is not guaranteed in M, the method that we indicated above may not be applied to explicitely construct such a code. The aim of Section 4 is to present a method of completion in the case where M is a submonoid of A^* with finite rank. Regular submonoids consitute certainly the most famous examples of such submonoids. In Section 3, we dry a preliminary study of the so-called notion of weak rank of a submonoid.

We assume the reader to be familiar with the basic notions concerning automata. Many famous open questions are concerned by synchronization in automata. Černý's conjecture [Pi 78], and the topic of the so-called synchronizing codes [BerP 85, p.115, 240, 331] [C 88] constitutes two classical examples.

Let $\mathcal{A}$ be a (non necessarily finite) deterministic trim automaton, with set of states Q. Given a word $w \in A^*$, we define its rank by the positive integer $r(w) = |Q.w|$. We set $r(\mathcal{A}) = min\{r(w)|w \in A^*, r(w) \geq 1\}$. It is important to note that with this definition, states $q \in Q$ may exist unless the transition $q.w$ is defined. In a natural way, the preceding definition may be extended to subsets of A^*: given a subset L of A^*, we set $r(L) = r(\mathcal{M})$, where $\mathcal{M}$ stands for the minimal deterministic trim automaton whith behavior L.

In the sequel, we consider the case where L is a submonoid M of A^*.
We denote by i the initial state, and by T the set of terminal states in the corresponding minimal trim automaton. We assume that $r(M) = r(\mathcal{M}) < \infty$. In particular, this condition is satisfied by regular submonoids. Let u be a word with minimal rank, and let $q \in Q$ such that $q.u \neq \emptyset$. Let us introduce some notation:

Notation 1
1) Since the automaton $\mathcal{M}$ is trim, a pair of words u', u" and a state $t_0 \in T$ exist such that $q = i.u'$, and $q.uu" = i.u'uu" = t_0$. We set:

$$y = u'uu" \tag{2}$$

By construction, we have $r(y) \leq r(u)$, thus $r(y) = r(u) = r(\mathcal{M})$.

2) We set:

$$Q_0 = Q.y, \quad T_0 = Q.y \cap T \tag{3}$$

By definition, we have $|T_0| \leq r(\mathcal{M})$.

3) Since A^* operates on Q, with each word $w \in A^*$, we associate the unique partial mapping $f_w : Q \longrightarrow Q$ that is defined by $f_w(q) = q.w$.

Lemma 4. *For any word $\alpha \in A^*$, if we have $\alpha y \in M$, then $f_{\alpha y}$ is a one-to-one total mapping onto Q_0.*

Proof of Lemma 4. 1) The first step consists in establishing that, given two states $q, q' \in Q_0$, for any word $w \in A^*$, we have $q.w \in Q$ iff $q'.w \in Q$. By contradiction, assume that a word $w \in A^*$ exists such that $q.w \in Q$ and $q'.w \notin Q$. Since we have $1 \leq |Q.yw| \leq r(\mathcal{M}) - 1$, we contradict the minimality of $r(y) = r(\mathcal{M})$.

2) Let $t_0 \in T_0 \subset Q_0$. Since we have $\alpha y \in M$, we have $t_0.\alpha y \neq \emptyset$ therefore, according to 1), the partial mapping $f_{\alpha y}$ is in fact defined for any state $q \in Q_0$. More precisely, we have $f_{\alpha y}(Q_0) \subset Q_0$.
3) Since $r(w) = r(\mathcal{M})$ is finite, $f_{\alpha y}$ is necessarily injective (otherwise, we have $r(y.\alpha y) = |Q_0.\alpha y| < |Q_0| = r(y)$). Consequently, since Q_0 is a finite set, $f_{\alpha y}$ is bijective.

■

Lemma 5. *We have* $|T_0|$*=1.*

Proof of Lemma 5. Let $q, q' \in T_0$, and let $\alpha, \beta \in A$ such that $q = i.\alpha y$, $q' = i.\beta y$. Without loss of generality we prove that, for any word $u \in A^*$, if $i.\alpha yu \in T$, then we have also $i.\beta yu \in T$. Assume that $(i.\alpha y).u \in T$, thus $\alpha yu \in M$. According to Lemma 4, and since Q_0 is a finite set, an integer n exists such that $(f_{\alpha y})^n = id_{Q_0}$. As a consequence we have: $(i.\beta y).u = (i.\beta y)(\alpha y)^n u = i.\beta y.(\alpha y)^{n-1}(\alpha yu)$, thus $(i.\beta y).u \in T$. Since $\mathcal{M}$ is a minimal automaton, the conclusion of Lemma 5 follows.

■

The following result constitutes an important step in the proof of Proposition 1:

Lemma 6. *For any word* $\alpha \in A^*$*, if* $\alpha y \in M$*, then for any integer* $n \geq 1$*, we have* $i.(\alpha y)^n = i.\alpha y$*.*

Proof of Lemma 6. According to Lemma 4, a (minimal) positive integer n exists such that $(f_{\alpha y})^n = id_{Q_0}$. If $n = 1$, then the result is trivial. Assume that $n \geq 2$. By contradiction, we assume that an integer $k \geq 2$ exists such that $i.(\alpha y)^k \neq i.\alpha y$, and we consider a word $u \in A^*$ such that we have $(i.\alpha y).u \in T$. Since we have $\alpha y, \alpha yu \in M$, we obtain:

$$i.(\alpha y)^k.u = i.(\alpha y)^{k-1}(\alpha yu) \in T$$

Conversely, let $u \in A^*$ such that we have $i.(\alpha y)^k u \in T$.
Since we have $\alpha y, (\alpha y)^k u \in M$ we obtain:

$$i.\alpha yu = i.\alpha y(\alpha y)^{kn}u = i.\alpha y(\alpha y)^{k(n-1)}.(\alpha y)^k u \in T$$

Since we assume that $i.\alpha y \neq i.(\alpha y)^k$, this contradicts the minimality of $\mathcal{M}$.

■

As a consequence of the preceding lemmas, we obtain the following result:

Proposition 1. *With the preceding notation, given a word* $y \in M \setminus F(X^*)$*, if* $r(y) = r(\mathcal{M})$ *then a unique state* t_0 *exists such that we have* $T_0 = T \cap Q.y = \{t_0\}$*. Moreover the four following properties hold:*
(i) For any word $\alpha \in A^*$*, if* $\alpha y \in M$*, then we have* $i.\alpha y = t_0.\alpha y = t_0$*.*
(ii) For any state q *of* $\mathcal{M}$*, a word* $\alpha \in A^*$ *exists such that* $q.\alpha y = t_0$*.*
(iii) For any word $\alpha \in A^*$*, if we have* $t_0.\alpha y = t_0$*, then the word* $y\alpha y$ *belongs to* M*.*
(iv) For any word $\alpha \in A^*$*, if the word* αy^2 *belongs to* M*, then the word* αy *belongs also to* M*.*

Property (iv) is a consequence of Lemma 4. With Property (iii), it constitutes a main step in the proofs of the results of Section 4.

4 A Method for Completing a Code in a Submonoid with Finite Rank

With the notation of Section 3, we consider a word $y \in M$ such that $r(y) = r(\mathcal{M})$. Let $X \subset M$ be a non weakly M-complete code, and let $t \in M \setminus F(X^*)$. Since we have $ty \in M \setminus F(X^*)$, according to Lemma 1, a word $z \in M$ and an integer k exist such that $w = z^k(ty)$ is a primitive word in $M \setminus F(X^*)$. Since we have also $w, w^2 \in A^*y \cap M$, we obtain:

$$r(w) = r(w^2) = r(\mathcal{M}) \quad \textit{and} \quad i.w = i.w^2 = t_0. \tag{4}$$

As a consequence, the results of Proposition 1 may be also formulated by substituting each of the words w, w^2 to y. More precisely:

Corollary 1. *With the preceding notation, the four following conditions hold:*
(i) For any word $\alpha \in A^*$, if we have $\alpha w \in M$, ($\alpha w^2 \in M$) then we have also, for any integer paar $n, p \geq 1$, $i.(\alpha w)^n = t_0.(\alpha w)^p = \{t_0\}$ ($i.(\alpha w^2)^n = t_0.(\alpha w^2)^p = \{t_0\}$).
(ii) For any state q of $\mathcal{M}$, a word $\alpha \in A^*$ exists such that, for any positive integer n, $q.\alpha w^n = \{t_0\}$.
(iii) For any word $\alpha \in A^*$, if we have $t_0.\alpha w = t_0$, ($t_0.\alpha w^2 = t_0$) then we have $w\alpha w \in M$ ($w^2\alpha w^2 \in M$).
(iv) For any word $\alpha \in A^*$, if we have $\alpha w^2 \in M$, then we have also $\alpha w \in M$.

Moreover, according to Lemma 2, w^2 is a $X^* \times A^*$ overlapping-free word. We set:

Notation 2
$\overline{y} = w^2$.
$N = (\overline{y}A^* \cap A^*\overline{y} \setminus A^*w\overline{y}) \cap M$
$Y = N \setminus (N \cup X)(N \cup X)^+$
$\hat{X} = X \cup Y$

By construction, we have $\varepsilon \notin N$, thus $\varepsilon \notin Y$. Moreover, by construction:

$$N \subset (X \cup Y)^* \tag{5}$$

Clearly, the inclusion is strict. Finally, we note that we have $\overline{y} \in Y$.

Proposition 2. *With the preceding notation, $\hat{X}$ is a code.*

Proof Sketch of Proposition 2. We consider an arbitrary equation between the words of $\hat{X}$. We note that, since X is a code, without loss of generality we may assume that at least one occurence of a word in Y occurs in at least one of the two sides of this equation. Moreover, since $\overline{y}$ is a factor of any word in Y no word in Y may be a factor of a word in X^*. More precisely, a pair of words $y_1, y'_1 \in Y$ exists such that our equation takes the form:

$$x_1y_1 \cdots x_ny_nx_{n+1} = x'_1y'_1 \cdots x'_my'_mx'_{m+1} \qquad x_i, x'_j \in X^* \quad y_i, y'_j \in Y \qquad (6)$$

Finally, without loss of generality, we assume that $|x'_1| \leq |x_1|$.
By definition, we have $\overline{y} \in P(y_1)$. Since we have $\overline{y} \notin F(X^*)$, and according to Lemma 2, we have in fact $x_1 = x'_1$.
Now, we consider the word $y_1 \in Y$. Once more without loss of generality, and by contradiction, we assume that $|y'_1| < |y_1|$. We examine the word $y'_1x'_2 \cdots x'_my'_mx'_{m+1}$. We set $y'_j = z_{2j-1}$ $(1 \leq j \leq m)$ and $x'_j = z_{2j-2}$ $(2 \leq j \leq m+1)$, and we denote by k the greatest positive integer such that we have $y_1 = z_1 \cdots z_k u$, with $z_k, u \neq \varepsilon$. Since we assume that $y_1 \in Y$, necessarily we have $u \notin (X \cup Y)^+$.
a) First, we assume that we have $z_k \in Y$. We compare the length of u with the four integers $i|w|$ $(0 \leq i \leq 3)$. In each of the corresponding cases, a combinatoric study leads to contradicts either the definition of Y, or the fact that w is a primitive word (cf (1)), or the fact that w cannot be a factor of a word in X^*. In particular, in the case where $|u| > 3|w|$, Property (iii) from Corollary 1 plays a promininent part to prove that $y_1 \in (X \cup Y)(X \cup Y)^+$, a contradiction with the definition of $Y \subset N$.
b) Assuming that we have $z_k \in X^*$, as in the preceding case (a), comparing the length of $|u|$ to $i|w|$ $(0 \leq i \leq 3)$ leads to similar contradictions. Finally, we obtain $y_1 = y'_1$. By induction, we conclude that Equation (6) is necessarily trivial.

■

Proposition 3. *With the preceding notation, $\hat{X}$ is weakly M-complete.*

Proof of Proposition 3. Since we have $N \subset (X \cup Y)^*$, it suffices to prove that any word $u \in M$ is a factor of a word in N.
Let $u \in M$. Set $u_0 = \overline{y}u\overline{y}$ and denote by i_0 the greatest positive integer such that w^{i_0} is a suffix of u_0 (by construction, we have $i_0 \geq 2$). If we have $i_0 = 2$, then by definition, we obtain $u_0 \in N$. Assume that $i_0 \geq 3$ and set $u'_0 = u_0w^{-i_0}$. Since we have $u_0 = u'_0w^{i_0-2}w^2 \in M$, according to Corollary 1 (iv), each of the words $u'_0w^{i_0-1}, u'_0w^{i_0-2}, \cdots, u'_0w^2$ belongs to M. Moreover, by construction we have $u'_0w^2 \in \overline{y}A^* \cap A^*\overline{y} \setminus A^*w\overline{y}$. By definition, this implies $u'_0 \in N$. As a consequence, we have $u_0 \in N.P(N)$, hence $u_0 \in P(N)$, thus $u \in F(N)$.

■

The following statement synthetises the results of the two preceding propositions:

Theorem 3. *Given a regular submonoid of A^*, each of the two following properties holds:*
(i) A regular weakly M-complete code exists.
(ii) Any code $X \subset M$ may be embedded in a weakly M-complete code $\hat{X}$, with respect to Notation 1. In particular, if X is regular, so is $\hat{X}$.

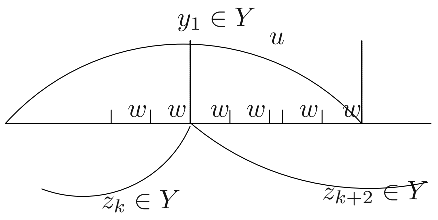


Fig. 1. Proof of Proposition 2: the case where $z_k \in Y$, with $|u| > 3|w|$. If $z_{k+1} \neq \varepsilon$, then by definition we have $|u| < |z_{k+1}|$, which contradicts $w \notin F(X^*)$. Consequently we obtain $z_{k+1} = \varepsilon$ which, according to Corollary 1 (iii), implies $u \in N$, thus $y_1 \in (X \cup Y)(X \cup Y)^+$.

Example 2. Let $A = \{a, b\}$, $M = \{a, ab, ba\}^*$, and $X = \{abaab\}$. In the minimal deterministic automaton with behavior M, we have $r(abba) = r(M) = 1$. With the previous notations, we have $T_0 = \{t_0\} = \{i\}$ (cf Figure 4). The word *bababa* is uncompletable in X^* and $\overline{y} = bababa.abba$ is a primitive element of M. This leads to compute the regular set $\hat{X}$ as indicated above.

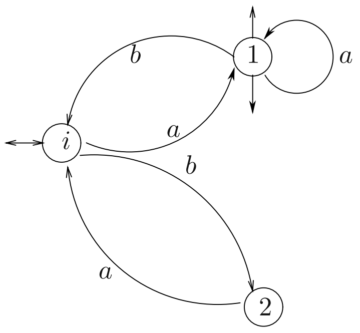


Fig. 2. Automata of M in Example 2: $Q.abba = \{i\}$

Conclusion

In [BlH 85], the authors establish, in a constructive way, the existence of a weakly M-complete code for an arbitrary submonoid of A^*. However, in any case the resulting set is weakly M-dense. Recall that, according to [BerP 85, p. 224], any regular set is very thin (i.e. $X^* \not\subset F(X)$) moreover, according to [NS 03], for an arbitrary subset of M, being very thin implies being strongly M-thin which implies M-thin. Consequently, Property (i) from Theorem 3 brings an affirmative

answer to the question of the existence of strongly M-thin weakly M-complete codes [NS 03].
We note also that many topics of theoretical computer science are concerned by our construction: in view of efficiently compute words of minimal rank, the framework of synchronizing words and collapsing words is illustrated by powerfull results [Pi 78, SaS 91] [ACV 03]. Clearly the computation of remarkable uncompletables words is also concerned [NS 01].
Finally, we would like to mention that the question of M-completion may be formulated for very classical and usefull classes of codes, such as prefix codes, codes with finite deciphering delay or circular codes.

References

[ACV 03] Ananichev D.S., Cherubini A. and M.V. Volkov Image reducing words and subgroups of free groups, *Theoret. Comput. Sci.* **307** (2003) 77-92.

[Bea 93] Béal M.-P., "Codage Symbolique", Masson, 1993.

[BerP 85] Berstel J., and D. Perrin, "Theory of Codes" Academic Press, 1985.

[BlH 85] Blanchard F and G. Hansel, systèmes codés, *Theoret. Comput. Sci.* **44** (1986) 17-49.

[Br 98] On maximal codes with bounded synchronizing delay, *Theoret. Comput. Sci* *204* (1998), 11-28.

[BrLa 96] Bruyère V. and M. Latteux, Variable-Length Maximal Codes, *Lecture Notes Comput. Sci.* **1099** (1996) 24-47.

[BrWZ 90] Bruyère V., Wang V. and L. Zhang, On completion of codes with deciphering delay, *European J. Combin.* **11** (1990), 513-521.

[C 88] Carpi A., On synchronizing unambigous automata *Theoret. Comput. Sci.* **60** (1988) 285-296.

[DeFR 85] De Felice C. and A. Restivo, Some results on finite maximal codes, *Theoret. Info. and Appl.* **19**, 4 (1985), 383-403.

[DevL 91] Devolder J. and I. Litovsky, Finitely generated bi ω-languages, *Theoret. Comput. Sci.* **85** (1991), 33-52.

[ER 83] Ehrenfeucht A and S. Rozenberg, Each regular code is included in a regular maximal one, *Theoret. Info. and Appl.* **20**, 1 (1985), 89-96.

[G 03] Guesnet Y., On maximal synchronous code, *Theoret. Comput. Sci.*, **307** (2003) 129-138.

[LaT 86] Latteux M. and E. Timmermann, Finitely generated ω-languages, *Info. Process. Letter* **23** (1986) 171-175.

[Li 91] Litovsky I., Prefix-free languages as ω-generators, Info. Process. Letters **37** (1991) 61-65.

[LiT 87] Litovsky I. and E. Timmerman, On generator of rationnal ω-languages, *Theoret. Comput. Sci.* **53** (1987) 187-200.

[Lo 83] Lothaire M., "Combinatorics on Words", Addison-Wesley, 1983.

[Nh 01] Nguyen Huong Lam, Finite maximal solid codes, *Theoret. Comput. Sci* **262** (2001) 333-347.

[NS 01] Néraud J and C. Selmi, On codes with a finite deciphering delay: constructing uncompletable words, *Theoret. Comput. Sci.* **255** (2001), 152-162.

[NS 02] Néraud J. and C. Selmi, Locally complete sets and finite decomposable codes, *Theoret. Comput. Sci.* **273** (2002) 185-196.

[NS 03] Néraud J. and C. Selmi, A characterization of complete finite prefix codes in arbitrary submonoids of A^*, *Int. Journ. of Alg. and Comp.* **13**, No 5 (2003) 507-516.

[NS 03A] Néraud J. and C. Selmi, Free monoid theory: Maximality and completeness in arbitrary submonoids, *Jour. of Aut. Langu. and Comb.*, to appear.

[Pi 78] Pin J.E., le problème de la synchronization. Contribution à l'étude de la conjecture de Černý, Thèse de 3em cycle, Paris (1978).

[R 89] Restivo A., Finitely generated sofic systems, *Theoret. Comput. Sci.* **65** (1989), 265-270.

[R 90] Restivo A., Codes and local constraints, *Theoret. Info. and Appl.* **72** (1990), 55-64.

[RSS 89] Restivo A., Salemi S. and T. Sportelli, Completing codes, *Theoret. Info. and Appl.* **23**, 2 (1989), 135-147.

[SaS 91] Sauer N. and M.G. Stone, Composing functions to reduce image size, *Ars. Combin.* **31** (1991) 171-176.

On Computational Universality in Language Equations

Alexander Okhotin

School of Computing, Queen's University, Kingston, Ontario, Canada K7L3N6
okhotin@cs.queensu.ca
http://www.cs.queensu.ca/home/okhotin/

Abstract. It has recently been shown that several computational models – trellis automata, recursive functions and Turing machines – admit characterization by resolved systems of language equations with different sets of language-theoretic operations. This paper investigates how simple the systems of equations from the computationally universal types could be while still retaining their universality. It is shown that resolved systems with two variables and two equations are as expressive as more complicated systems, while one-variable equations are "almost" as expressive. Additionally, language equations with added quotient with regular languages are shown to be able to denote every arithmetical set.

1 Introduction

Language equations have been studied since the 1960s, when both finite automata [17] and context-free grammars [6,1] were given algebraic characterizations by resolved systems of language equations with union and concatenation (one-sided in the case of finite automata). The following decades saw a certain decline of interest in this mathematical object. Recently the interest in language equations has renewed, with new results related to applied logic [2,19], biocomputing [7], language specification [12], systems design [18], as well as theoretical work in the area [8,10,13].

One direction of theoretical research concerns characterizing models of computation by language equations. Recently, trellis automata [4], a model of parallel computation, were shown to be equivalent to resolved systems of language equations with union, intersection and linear concatenation [14], and thus to a class of transformational grammars called linear conjunctive grammars [11,12]. Recursive languages were characterized by unique solutions of systems with union, intersection, complement and concatenation [13], while least and greatest solutions of these systems define exactly the recursively enumerable and the co-recursively enumerable sets, respectively.

In light of the recent research on minimal universal Turing machines [9] and programmed grammars [5], the computational universality demonstrated by language equations naturally raises the question of how complicated must a system of language equations be in order to be able to denote every language from one of these classes? How complicated a system has to be in order to simulate any particular universal Turing machine?

For much weaker types of language equations, those equivalent to trellis automata, it has recently been proved that two equations with two variables are always enough, while a single one-variable equation can denote a language closely related to any given trellis language [15]. For the computationally universal language equations, the original proof [13] yields systems of at least ten equations. This paper aims to improve this result, using the one-nonterminal representation of trellis automata [15] as the technical foundation.

2 Language Equations

Definition 1 (System of language equations). *Fix a finite set of language-theoretic operations, typically containing at least union and linear concatenation. Let Σ be an alphabet. Let $n \geqslant 1$. Let $X = (X_1, \ldots, X_n)$ be a set of language variables, which assume values of languages over Σ. Let $\varphi_1, \ldots, \varphi_n$ be expressions that depend upon the variables X and may contain these variables, the constant languages $\{\varepsilon\}$ and $\{a\}$ (for all $a \in \Sigma$) and language-theoretic operations chosen above. Then*

$$\begin{cases} X_1 = \varphi_1(X_1, \ldots, X_n) \\ \quad\vdots \\ X_n = \varphi_n(X_1, \ldots, X_n) \end{cases} \tag{1}$$

is called a resolved system of equations over Σ in variables X. A vector of languages $L = (L_1, \ldots, L_n)$ is a solution of (1) if for every i the value of φ_i under the assignment $X_1 = L_1, \ldots, X_n = L_n$ is L_i.

A system may have no solutions, a unique solution, or multiple solutions. *Least* and *greatest* solutions under the partial order of componentwise inclusion are often considered. If all operations used in the right-hand sides of equations in (1) are monotone (e.g., union, intersection, concatenation, quotient, homomorphism and so on, but *not* complement), then least and greatest solutions are guaranteed to exist by a straightforward argument based upon the lattice-theoretic fixed point theorem [1,12]. Least solutions are often used as a semantics of such language equations [1,12] because of the natural correspondence to the derivability in formal grammars.

Systems of language equations with union and unrestricted concatenation are equivalent to context-free grammars in expressive power [6,1], and similarly systems of language equations with union and linear concatenation are equivalent to linear context-free grammars. It has recently been shown that systems of language equations with union, intersection and concatenation are equivalent to *conjunctive grammars* [11], which are context-free grammars with an explicit intersection operation in the formalism of rules and with the machinery of derivation modified to implement this operation [12]. Systems of language equations with union, intersection and linear concatenation have been proved to be equivalent to *trellis automata* [4,12,14], a model of parallel computation, and to *linear conjunctive grammars* [11].

The generative power of all of the mentioned classes does not go outside context-sensitive languages. However, language equations with concatenation, union, intersection *and complement* demonstrate computational universality [13] in the sense that unique (least, greatest, resp.) solutions of systems of such equations can denote exactly the recursive (recursively enumerable, co-recursively enumerable, resp.) sets. This paper presents a refinement of these results, representing arbitrary recursive, r.e. and co-r.e. sets with language equations of much simplified structure. Trellis automata [4], their recently found representation by language equations with union, intersection and linear concatenation [14] and the related descriptional complexity results [15] will be essentially used for that purpose. Let us give a short introduction to these concepts.

3 Trellis Automata and Their Algebraic Characterization

(Systolic) trellis automata [4] were introduced in early 1980s as a model of a massively parallel system with simple identical processors connected in a uniform pattern. *Homogeneous trellis automata* are a particular case of trellis automata, in which the connections between nodes form a figure of triangular shape, as shown in Figure 1. These automata are used as acceptors of strings loaded from the bottom, and the acceptance is determined by the topmost element. In the following they will be referred to as just *trellis automata*.

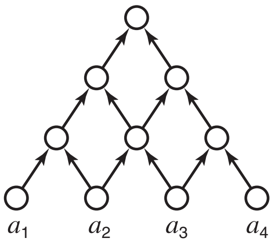


Fig. 1. Computation of a trellis automaton.

Following the notation of [14], define a trellis automaton as a quintuple $M = (\Sigma, Q, I, \delta, F)$, where Σ is the input alphabet, Q is a finite nonempty set of states (of processing units), $I : \Sigma \to Q$ is a function that sets the initial states (loads values into the bottom processors), $\delta : Q \times Q \to Q$ is the transition function (the function computed by processors) and $F \subseteq Q$ is the set of final states (effective in the top processor). Given a string $a_1 \ldots a_n$ ($a_i \in \Sigma$, $n \geqslant 1$), every node corresponds to a certain substring $a_i \ldots a_j$ ($1 \leqslant i \leqslant j \leqslant n$) of symbols on which its value depends. The value of a bottom node corresponding to one symbol of the input is $I(a_i)$; the value of a successor of two nodes is δ of the values of these ancestors. Denote the value of a node corresponding to $a_i \ldots a_j$ as $\Delta(I(a_i \ldots a_j)) \in Q$: here $I(a_i \ldots a_j)$ is a string of states (the bottom row of the trellis), while Δ denotes the result (a single state) of a triangular computation starting from a row of states. By definition, $\Delta(I(a_i)) = I(a_i)$ and $\Delta(I(a_i \ldots a_j)) = \delta(\Delta(I(a_i \ldots a_{j-1})), \Delta(I(a_{i+1} \ldots a_j)))$. The language recognized by the automaton is defined as $L(M) = \{w \mid \Delta(I(w)) \in F\}$.

The computational equivalence of trellis automata to language equations over $\{\cup, \cap, \text{LIN}\cdot\}$ follows from the following two theorems:

Theorem 1 ([14]). *For every system of language equations $X_i = \varphi_i(X_1, \ldots, X_m)$ ($1 \leqslant i \leqslant m$) with union, intersection and linear concatenation, such that L is the first component of its least solution, there exists and can be effectively constructed a trellis automaton M, such that $L(M) = L \setminus \{\varepsilon\}$.*

The proof of Theorem 1 essentially relies on the normal-form theorem for linear conjunctive grammars [11] and then on subset construction [14]. Every state of the constructed trellis automaton represents a set of nonterminals of a normal-form linear conjunctive grammar, or, equivalently, a set of variables of a system of language equations with restricted right-hand sides.

Theorem 2 ([14]). *For every trellis automaton $M = (\Sigma, Q, I, \delta, F)$, there exists and can be effectively constructed a system of language equations $X_i = \varphi_i(X_1, \ldots, X_m)$ ($1 \leqslant i \leqslant m$) with union, intersection and linear concatenation, such that the first component of its unique solution is $L(M)$.*

Theorem 2 can be proved by taking the set of variables $\{S\} \cup \{X_q \mid q \in Q\}$, and then specifying the equation $S = \bigcup_{q \in F} X_q$ for the first variable, and the equation $X_q = \bigcup_{\substack{a \in \Sigma: \\ I(a)=q}} a \ \cup \ \bigcup_{\substack{q_1, q_2 \in Q: \\ \delta(q_1,q_2)=q}} \bigcup_{b,c \in \Sigma} \big(bX_{q_2} \cap X_{q_1}c\big)$ for the variables corresponding to the states of the automaton.

An important property of this language family is that it contains the language of valid accepting computations of any Turing machine. Indeed, it is well-known [3] that for every Turing machine T over an alphabet Σ each of the following two languages is an intersection of two linear context-free languages

$$L_{\text{Acc.Comp.}T} = \{w\natural C_T(w) \mid T \text{ halts on } w \text{ and accepts}\} \subseteq (\Sigma \cup \{\natural\} \cup \Gamma)^* \quad \text{(2a)}$$
$$L_{\text{Rej.Comp.}T} = \{w\sharp C_T(w) \mid T \text{ halts on } w \text{ and rejects}\} \subseteq (\Sigma \cup \{\sharp\} \cup \Gamma)^* \quad \text{(2b)}$$

for a suitable encoding $C_T : \Sigma^* \to \Gamma^*$ of a computations of T. This immediately gives a linear conjunctive grammar for each of these languages [11], and hence a trellis automaton or a system of language equations with union, intersection and linear concatenation.

Let us state another result on this language family which provides the technical foundation for the constructions of this paper. Using the trellis automaton representation, it was shown that the hierarchy of languages denoted by systems of n equations over $\{\cup, \cap, \text{LIN}\cdot\}$ collapses, and every trellis language can be denoted by a system of just two equations [15]. Given a trellis automaton M with n states $\{q_1, \ldots, q_n\}$, let $d = 6n - 1$ and consider the following *auxiliary language*:

$$L'_M = L_{control} \cup L_{left} \cup L_{right}, \quad \text{where} \tag{3a}$$
$$L_{control} = \{w \mid |w| = 1 \ (\mathrm{mod}\ d)\}, \tag{3b}$$
$$L_{left} = \{xw \mid |w| = 1 \ (\mathrm{mod}\ d), |x| = 2n + 2i - 1, \text{where } \Delta(I(w)) = q_i\}, \tag{3c}$$
$$L_{right} = \{wy \mid |w| = 1 \ (\mathrm{mod}\ d), |y| = 2i - 1, \text{where } \Delta(I(w)) = q_i\} \tag{3d}$$

Theorem 3 ([15]). *Let Σ be an alphabet, let $\$ \notin \Sigma$. Then for every trellis automaton M there exist and can be effectively constructed:*

1. *a one-variable resolved language equation of the form $X = \xi(X)$, where ξ uses the operations of union, intersection and linear concatenation, that has unique solution $L(M)\$ \cup L'_M$.*
2. *a resolved system of two language equations, $X = \psi(Y)$ and $Y = \varphi(Y)$, where φ and ψ contain union, intersection and linear concatenation, which has unique solution $(L(M), L'_M)$.*

Now everything is ready for constructing small computationally universal language equations, which demonstrate the same expressive power as the systems with quite a few variables constructed in a recent paper [13].

4 Constructing a Universal Language Equation

Theorem 4. *For every Turing machine T over an alphabet Σ there exist and can be effectively constructed an alphabet $\Sigma' \supset \Sigma$ and a language equation $X = \varphi(X)$ over Σ' (where φ uses union, intersection, complement and concatenation), which has the* ***least*** *solution $\dagger L(T) \cup L'$ for some $L' \subseteq (\Sigma' \setminus \{\dagger\})^*$.*

Proof. Consider the language $L_{\mathrm{Acc.Comp.}T} \subseteq \Sigma^* \natural \Gamma^*$ (2a) of accepting computations of T, where Γ is some alphabet used to represent these computations. $L_{\mathrm{Acc.Comp.}T}$ is linear conjunctive, and hence there exists a trellis automaton M that accepts it [14].

Next, consider the auxiliary language L'_M (3) which encodes the computations of M. By Theorem 3, there exists a language equation $X = \xi(X)$ over the alphabet $\Sigma \cup \Gamma \cup \{\natural, \$\}$ that has the unique solution $L(M)\$ \cup L'_M$.

Consider the alphabet $\Sigma \cup \Gamma \cup \{\natural, \$, \dagger\}$ and construct the language equation

$$X = \xi\big(X \cap (\Sigma \cup \Gamma \cup \{\natural, \$\})^*\big) \cup \big(X \cap \dagger\Sigma^*\big) \tag{4}$$

Note that the language $(\Sigma \cup \Gamma \cup \{\natural, \$\})^*$ used in (4) is a regular language of extended star height 0, and hence could be represented as required by Definition 1 (in this case, e.g., as $\overline{(\varepsilon \cup \overline{\varepsilon})\dagger(\varepsilon \cup \overline{\varepsilon})}$). Such representations will not be given in the following, since they are not very readable, and at the same time can easily be constructed if needed.

Let us prove that the set of solutions of (4) is

$$\{\dagger L \cup L' \mid L \subseteq \Sigma^*, L' \subseteq (\Sigma \cup \Gamma \cup \{\natural, \$\})^* \text{ is a solution of } X = \xi(X)\} \tag{5}$$

If L' is a solution of $X = \xi(X)$ and $L \subseteq \Sigma^*$, then $\xi\big((\dagger L \cup L') \cap (\Sigma \cup \Gamma \cup \{\natural, \$\})^*\big) = \xi(L') = L'$, while $(\dagger L \cup L') \cap \dagger\Sigma^* = \dagger L$. Hence, the union of these two sets is $L' \cup \dagger L$, showing that this is a solution of (4).

Conversely, if a language L_0 satisfies (4), then

$$L_0 = L' \cup \dagger L, \tag{6}$$

where $L \subseteq \Sigma^*$ and

$$L' = \xi(L_0 \cap (\Sigma \cup \Gamma \cup \{\natural, \$\})^*) \tag{7}$$

Let us intersect both sides of (6) with $(\Sigma \cup \Gamma \cup \{\natural, \$\})^*$, obtaining

$$L_0 \cap (\Sigma \cup \Gamma \cup \{\natural, \$\})^* = (L' \cup \dagger L) \cap (\Sigma \cup \Gamma \cup \{\natural, \$\})^* = L' \cap (\Sigma \cup \Gamma \cup \{\natural, \$\})^* \tag{8}$$

Now note that $L' \subseteq (\Sigma \cup \Gamma \cup \{\natural, \$\})^*$ by (7) and by the monotonicity of ξ, which prevents the symbol $\dagger$ from appearing in the strings from $\xi(L_0 \cap (\Sigma \cup \Gamma \cup \{\natural, \$\})^*)$. Hence (8) can be simplified to $L_0 \cap (\Sigma \cup \Gamma \cup \{\natural, \$\})^* = L'$. Substituting the latter for the argument of ξ in (7), we obtain that $L' = \xi(L')$, or L' is a solution of $X = \xi(X)$. Therefore, L_0 is in (5).

Since $X = \xi(X)$ is known to have a unique solution $L(M)\$ \cup L'_M$, (5) can be rewritten as

$$\{\dagger L \cup L(M)\$ \cup L'_M \mid L \subseteq \Sigma^*\}, \tag{9}$$

which is hence the set of solutions of (4).

Now let us construct a new language equation for the same variable X to be used in conjunction with (4), which will require that for every string w accepted by T the string $\dagger w$ should be in X. This requirement is directly specified by this inclusion [13]:

$$\dagger L_{\text{Acc.Comp.}T} \subseteq X\natural\Gamma^* \tag{10}$$

Recalling that $L_{\text{Acc.Comp.}T} = L(M)$, and right-concatenating \$ to both sides of (10), this inclusion can be equivalently rewritten as

$$\dagger L(M)\$ \subseteq X\natural\Gamma^*\$ \tag{11}$$

Since every solution (9) of the equation (4) yields $L(M)\$$ when intersected with $(\Sigma \cup \Gamma \cup \{\natural, \$\})^*\$$, (11) can be rewritten as an inclusion

$$\dagger\big(X \cap (\Sigma \cup \Gamma \cup \{\natural, \$\})^*\$\big) \subseteq X\natural\Gamma^*\$ \tag{12}$$

or as an implicit equation

$$\dagger\big(X \cap (\Sigma \cup \Gamma \cup \{\natural, \$\})^*\$\big) \cap \overline{X\natural\Gamma^*\$} = \varnothing \tag{13}$$

The system comprised of a resolved equation (4) and an implicit equation (13) has the set of solutions

$$\{\dagger L \cup L(M)\$ \cup L'_M \mid L(T) \subseteq L \subseteq \Sigma^*\}, \tag{14}$$

and the least among these solutions is $\dagger L(T) \cup L(M)\$ \cup L'_M$.

The final step of the argument is to transcribe the system (4,13) as a single resolved language equation, which will retain the set of solutions (14). Let us prove this as an abstract claim, which will be reused in the subsequent proofs.

Claim 1 *Let A be an alphabet, let $¢ \notin A$. Let $\eta(X)$ and $\theta(X)$ be formulae, such that $\eta(L') \subseteq A^*$ and $\theta(L') \subseteq A^*$ for every $L' \subseteq (A \cup \{¢\})^*$. Then*

$$L = \eta(L) \tag{15a}$$

$$\theta(L) = \varnothing \tag{15b}$$

if and only if

$$L = \eta(L) \cup \big(\overline{L} \cap ¢\theta(L)\big) \tag{16}$$

Proof. $\Rightarrow$ Using (15b), rewrite the right-hand side of (16) as $L \cup (\overline{L} \cap ¢\varnothing) = L \cup \varnothing = L$, and thus (16) holds.
$\Leftarrow$ Let L be a language that satisfies (16) and suppose there exists a string $w \in \theta(L)$. Then $¢w \in L$ if and only if $¢w \in \overline{L} \cap ¢\theta(L)$ (by (16) and by $\eta(L) \subseteq A^*$), which holds if and only if $¢w \notin L$ and $¢w \in ¢\theta(L)$, which is equivalent to $¢w \notin L$ and yields a contradiction. Hence, $w \notin \theta(L)$ for every $w \in A^*$, which means (15b).
Then (16) degrades to $L = \eta(L)$, proving (15a) as well. □

Resuming the proof of Theorem 4, let $A = \Sigma \cup \Gamma \cup \{\natural, \$, \dagger\}$, let

$$\eta(X) = \xi\big(X \cap (\Sigma \cup \Gamma \cup \{\natural, \$\})^*\big) \;\cup\; \big(X \cap \dagger\Sigma^*\big) \quad \text{and let} \tag{17a}$$

$$\theta(X) = \dagger\big(X \cap (\Sigma \cup \Gamma \cup \{\natural, \$\})^*\$\big) \cap \overline{X\natural\Gamma^*\$} \tag{17b}$$

Thus the system (4,13) has been represented in the form

$$X = \eta(X) \tag{18a}$$

$$\theta(X) = \varnothing \tag{18b}$$

Since η and θ satisfy the conditions of Claim 1, this claim states the equivalence of (18) to $X = \eta(X) \cup \big(\overline{X} \cap ¢\theta(X)\big)$, or, in the original notation,

$$X = \xi\big(X \cap (\Sigma \cup \Gamma \cup \{\natural, \$\})^*\big) \cup (X \cap \dagger\Sigma^*) \cup \overline{X} \cap ¢\left(\dagger\big(X \cap (\Sigma \cup \Gamma \cup \{\natural, \$\})^*\$\big) \cap \overline{X\natural\Gamma^*\$}\right)$$

where the formula ξ is given by Theorem 3. Being equivalent to the system (4,13), this equation has the set of solutions (14), and hence

$$\dagger L(T) \cup L(M)\$ \cup L'_M \tag{19}$$

is its least solution that satisfies the statement of the theorem, which is easily seen by assuming $\Sigma' = \Sigma \cup \Gamma \cup \{\natural, \$, \dagger, ¢\}$ and $L' = L(M)\$ \cup L'_M$. □

Note that the least solution (19) of the constructed equation incorporates (a) the language recognized by the given Turing machine T, (b) the language of accepting computations of this Turing machine, recognized by a trellis automaton M constructed with respect to T, and finally (c) the language (3) that encodes the operation of M, which in turn encodes the operation of T.

Since the equation uses only one variable, these two layers of "junk" left by a double simulation cannot be stored anywhere else, and obtaining precisely $L(T)$ as a least solution of a single-variable language equation does not seem to be possible using the suggested method. It is conjectured that some recursively enumerable languages, including even quite simple ones, cannot be specified by a single-variable language equation at all:

Conjecture 1. There exists no language equation $X = \varphi(X)$ with set-theoretic operations and concatenation, such that $\{a^n b^n c^n \mid n \geqslant 0\}$ is its unique, least or greatest solution.

If more than one variable might be required to denote some languages precisely, the question is, how many? As in the case of language equations with union, intersection and linear concatenation [15] (equivalent to trellis automata and linear conjunctive grammars), there is no infinite hierarchy of n-variable languages, as two variables turn out to be always sufficient:

Theorem 5. *For every Turing machine T over an alphabet Σ there exist and can be effectively constructed a resolved system of language equations of the form*

$$Y = Y \tag{20a}$$
$$X = \psi(X) \tag{20b}$$

with set-theoretic operations and concatenation, which has the least solution $(L(T), \dagger L(T) \cup L')$ for some $L' \subseteq (\Sigma' \setminus \{\dagger\})^*$.

Proof. Let $X = \varphi(X)$ be the equation given by Theorem 4, and consider the inclusion $X \cap \dagger\Sigma^* \subseteq \dagger Y$. Together, they can be represented in the form

$$X = \varphi(X) \tag{21a}$$
$$X \cap \dagger\Sigma^* \cap \overline{\dagger Y} = \varnothing \tag{21b}$$

The implicit equation (21b) specifies that for every string $\dagger w \in X$, the string w should be in Y. Hence, the least solution of (21) is, as requested, $(L(T), \dagger L(T) \cup L')$.

It remains to represent (21) in the form of a single equation $X = \psi(X)$, which can be done according to Claim 1. Coupled with a dummy equation (20a), this yields the resolved system (20) with the required least solution. $\square$

Theorems 4 and 5 claim the constructibility of a class of language equations from a class of machines. Another noteworthy fact is the existence of one particular resolved one-variable language equation, which demonstrates computational universality:

Proposition 1. *Let T be a universal Turing machine over $\{0, 1\}$. Then the language equation constructed for T by Theorem 4 can be called a universal language equation.*

Similarly to Theorem 4, co-recursively enumerable sets can be represented by greatest solutions of one-variable language equations.

Theorem 6. *For every TM T over an alphabet Σ there exist and can be effectively constructed an alphabet $\Sigma' \supset \Sigma$ and a language equation $X = \varphi(X)$ over Σ' (where φ uses union, intersection, complement and concatenation), which has the* ***greatest*** *solution $\dagger(\Sigma^* \setminus L(T)) \cup L'$ for some $L' \subseteq (\Sigma' \setminus \dagger)^*$.*

Proof (A sketch). Theorem 6 is proved very similarly to Theorem 4: first, a trellis automaton M for $L_{\text{Acc.Comp.}T}$ and a language equation $X = \xi(X)$ with the unique solution $L(M)\$ \cup L'_M$ are constructed. It is then transformed to an equation (4) over $\Sigma \cup \Gamma \cup \{\natural, \$, \dagger\}$, which has the set of solutions (9).

Then the additional condition that for every string w accepted by T the string $\dagger w$ should *not* be in X is specified (cf. the corresponding passage in the proof of Theorem 4), which is done by the following inclusion [13]:

$$\dagger L_{\text{Acc.Comp.}T} \subseteq \overline{X}\natural\Gamma^*, \tag{22}$$

This inclusion is then reformulated as an implicit equation in the same way as (10) is rewritten as (13) in the proof of Theorem 4.

Using Claim 1, this pair of a resolved equation and an implicit equation is converted to a single equation over the alphabet $\Sigma \cup \Gamma \cup \{\natural, \$, \dagger, ¢\}$. It has the set of solutions

$$\{\dagger L \cup L(M)\$ \cup L'_M \mid L \subseteq \Sigma^* \setminus L(T)\}, \tag{23}$$

and the greatest among them is $\dagger\overline{L(T)} \cup L(M)\$ \cup L'_M$. □

In order to characterize recursive sets by one-variable language equations, let us use the languages of accepting and rejecting computations of a single Turing machine that halts on every input.

Theorem 7. *For every TM T over an alphabet Σ that halts on every input there exist and can be effectively constructed an alphabet $\Sigma' \supset \Sigma$ and a language equation $X = \varphi(X)$ over Σ' (where φ uses union, intersection, complement and concatenation), which has the* ***unique*** *solution $\dagger L(T) \cup L'$ for some $L' \subseteq (\Sigma' \setminus \dagger)^*$.*

Proof. The proof is slightly more complicated than the similar proofs of Theorems 4 and 6. Given a Turing machine T that halts on every input, both the languages $L_{\text{Acc.Comp.}T} \subseteq \Sigma^*\natural\Gamma^*$ (2a) and $L_{\text{Rej.Comp.}T} \subseteq \Sigma^*\sharp\Gamma^*$ (2b) will be used.

Each of these languages is linear conjunctive. Therefore, their *union*, $L_{\text{Acc.Comp.}T} \cup L_{\text{Rej.Comp.}T}$, is also linear conjunctive, and hence there exists a trellis automaton M that recognizes this union. Consider the auxiliary language L'_M (3) defined with respect to this trellis automaton, and the language equation $X = \xi(X)$ over the alphabet $\Sigma \cup \Gamma \cup \{\natural, \sharp, \$\}$ with the unique solution $L(M)\$ \cup L'_M = L_{\text{Acc.Comp.}T}\$ \cup L_{\text{Rej.Comp.}T}\$ \cup L'_M$, which can be constructed according to Theorem 3.

Transform it to the equation

$$X = \xi\big(X \cap (\Sigma \cup \Gamma \cup \{\natural, \$\})^*\big) \;\cup\; \big(X \cap \dagger\Sigma^*\big) \tag{24}$$

over $\Sigma \cup \Gamma \cup \{\natural, \sharp, \$, \dagger\}$, which has the set of solutions

$$\{\dagger L \cup L(M)\$ \cup L'_M \mid L \subseteq \Sigma^*\}, \tag{25}$$

Now the following two conditions have to be specified: first, for every string $w \in \Sigma^*$ accepted by T the string $\dagger w$ should be in X; second, for every string $w \in \Sigma^*$ rejected by T the string $\dagger w$ should not be in X. Since every string in Σ^* is either accepted or rejected by T by assumption, this would completely define L in (25), thus making the solution unique.

These conditions can be expressed by the following two inclusions [13]:

$$\dagger L_{\text{Acc.Comp.}T} \subseteq X \natural \Gamma^* \tag{26a}$$

$$\dagger L_{\text{Rej.Comp.}T} \subseteq \overline{X} \sharp \Gamma^* \tag{26b}$$

As in the proof of Theorem 4, these inclusions can be rewritten in the form

$$\dagger\big(X \cap (\Sigma \cup \Gamma \cup \{\natural, \$\})^* \$\big) \cap \overline{X \natural \Gamma^* \$} = \varnothing \tag{27a}$$

$$\dagger\big(X \cap (\Sigma \cup \Gamma \cup \{\sharp, \$\})^* \$\big) \cap \overline{\overline{X} \sharp \Gamma^* \$} = \varnothing, \tag{27b}$$

or as a single implicit equation of this type:

$$\left(\dagger\big(X \cap (\Sigma \cup \Gamma \cup \{\natural, \$\})^* \$\big) \cap \overline{X \natural \Gamma^* \$}\right) \cup \left(\dagger\big(X \cap (\Sigma \cup \Gamma \cup \{\sharp, \$\})^* \$\big) \cap \overline{\overline{X} \sharp \Gamma^* \$}\right) = \varnothing \tag{28}$$

Using Claim 1, the pair of (24) and (28) can be rewritten as a resolved equation $X = \varphi(X)$ over the alphabet $\Sigma \cup \Gamma \cup \{\natural, \sharp, \$, \dagger, ¢\}$, which has the unique solution $\dagger L(T) \cup L(M)\$ \cup L'_M$ that satisfies the statement of the theorem. □

5 Universality Results for Extended Classes of Language Equations

Language equations with set-theoretic operations and concatenation can already denote all recursive sets by their unique solutions. It is interesting to observe that increasing the descriptive means of these language equations even slightly further may explode their expressive power beyond expectations. Consider adding the operation of quotient with regular languages (which, for instance, preserves the class of context-free languages, and the class of recursively enumerable languages as well), and behold universality of quite an unexpected kind:

Theorem 8. *For every arithmetical set $L \subseteq \Sigma^*$ [16] there exists a one-variable resolved language equation $X = \varphi(X)$ with concatenation, set-theoretic operations and quotient with regular languages, such that the unique solution of this equation is $\ddagger L \cup L'$ for some $L' \subseteq (\Sigma' \setminus \{\ddagger\})^*$.*

Proof. A language is in the arithmetical hierarchy if and only if it can be represented in the following predicate form [16]:

$$\{w \mid Q_1 x_1\, Q_2 x_2\, \ldots\, Q_n x_n : R(w, x_1, \ldots, x_n)\} \quad (Q_1, \ldots, Q_n \in \{\exists, \forall\}), \tag{29}$$

where $w, x_1, \ldots, x_n$ are assumed to be strings over Σ, and R is an $(n+1)$-ary recursive predicate. Let us represent these $(n+1)$-tuples of strings as strings, using the flat sign $\flat$ as a separator. Let T is a Turing machine that halts on every input and decides the predicate R under this encoding; $L(T) \subseteq (\Sigma \cup \{\flat\})^*$. Then (29) can be rewritten as follows:

$$\{w \mid Q_1 x_1\, Q_2 x_2\, \ldots\, Q_n x_n : w\flat x_1 \flat \ldots \flat x_n \in L(T)\} \quad (Q_1, \ldots, Q_n \in \{\exists, \forall\}) \tag{30}$$

By Theorem 7, there exists a language equation $X = \varphi(X)$ over an alphabet $\Sigma' \supset \Sigma \cup \{\flat, \dagger\}$, such that $\dagger L(T) \cup L'$ is its unique solution (for some $L' \subseteq (\Sigma' \setminus \{\dagger\})^*$). Let us construct an expression $\psi(X)$ that uses the operation of quotient with regular languages, such that $\psi(\dagger L(T) \cup L')$ would equal the set (30). This can be done inductively by representing each language

$$L_k = \{w\flat x_1 \flat \ldots \flat x_k \mid Q_{k+1} x_{k+1} : \ \ldots\ Q_n x_n\ w\flat x_1 \flat \ldots \flat x_n \in L(T)\} \tag{31}$$

as $\psi_k(\dagger L(T) \cup L')$. The basis, $k = n$, is clear: $L_n = L(T) = \{\dagger\}^{-1} \cdot (\dagger L(T) \cup L')$, hence $\psi_n(X) = \{\dagger\}^{-1} \cdot X$. The induction step, $k+1 \to k$, is based on the identity

$$L_k = \{w\flat x_1 \flat \ldots \flat x_k \mid Q_{k+1} x_{k+1} : \ w\flat x_1 \flat \ldots \flat x_k \flat x_{k+1} \in L_{k+1}\} \tag{32}$$

The cases of an existential and a universal quantifier are treated differently:

– If $Q_{k+1} = \exists$, then (32) can be simply expressed as the quotient $L_k = L_{k+1} \cdot (\flat \Sigma^*)^{-1}$. Define
$$\psi_k(X) = \psi_{k+1}(X) \cdot (\flat \Sigma^*)^{-1} \tag{33}$$

– If $Q_{k+1} = \forall$, the duality of the universal quantifier to the existential quantifier can be used to obtain the following representation: $L_k = \{w\flat x_1 \flat \ldots \flat x_k \mid \nexists x_{k+1} : \ w\flat x_1 \flat \ldots \flat x_k \flat x_{k+1} \notin L_{k+1}\}$. Then $L_k = \overline{L_{k+1}} \cdot (\flat \Sigma^*)^{-1} \cap \Sigma^* (\flat \Sigma^*)^k$. Define
$$\psi_k(X) = \overline{\psi_{k+1}(X)} \cdot (\flat \Sigma^*)^{-1} \cap \Sigma^* (\flat \Sigma^*)^k \tag{34}$$

Now $\psi_0(\dagger L(T) \cup L')$ gives the required representation of (30). It is left to combine the equation $X = \varphi(X)$ and the formula ψ_0 into a single equation. This can be done as follows:
$$X = \varphi(X \cap \Sigma'^*) \cup \ddagger \psi_0(X \cap \Sigma'^*), \tag{35}$$

where ψ_0 has been defined above, while φ comes from Theorem 7. The equation (35) has the unique solution $\ddagger L \cup \dagger L(T) \cup L'$, where L is the given arbitrary arithmetical set. □

In full, (35) is
$$\ddagger L \cup \dagger L(T) \cup L(M)\$ \cup L'_M, \tag{36}$$

a fruit of a triple simulation: the language L is obtained from the recursive language $L(T)$ by quantification (Theorem 8), $L(T)$ comes out of the trellis language $L(M)$ that encodes the accepting and the rejecting computations of T (Theorem 7), while $L(M)$ has its origin in the language L'_M of encoded computations of M (Theorem 3).

These results show that resolved language equations with one variable demonstrate various types of computational universality, which contributes to the theory of language equations and establishes one more connection to the computation theory.

References

1. J. Autebert, J. Berstel, L. Boasson, "Context-free languages and pushdown automata", in: Rozenberg, Salomaa (Eds.), *Handbook of Formal Languages*, Vol. 1, Springer-Verlag, Berlin, 1997, 111–174.
2. F. Baader, R. Küsters, "Solving linear equations over regular languages", *Proceedings of the 15th International Workshop on Unification* (UNIF 2001, June 18-19, 2001, Siena, Italy), 27–31.
3. B. S. Baker, R. V. Book, "Reversal-bounded multipushdown machines", *Journal of Computer and System Sciences*, 8 (1974), 315–332.
4. K. Culik II, J. Gruska, A. Salomaa, "Systolic trellis automata", I and II, *International Journal of Computer Mathematics*, 15 (1984), 195–212; 16 (1984), 3–22.
5. H. Fernau, "Nonterminal complexity of programmed grammars", *Theoretical Computer Science*, 296 (2003), 225–251; also in MCU 2001.
6. S. Ginsburg, H. G. Rice, "Two families of languages related to ALGOL", *Journal of the ACM*, 9 (1962), 350–371.
7. L. Kari, G. Thierrin, "Maximal and minimal solutions to language equations", *Journal of Computer and System Sciences*, 53:3 (1996), 487–496.
8. J. Karhumäki, I. Petre, "Conway's problem for three-word sets", *Theoretical Computer Science*, 289 (2002), 705–725.
9. M. Kudlek, Yu. Rogozhin, "A Universal Turing Machine with 3 States and 9 Symbols", *Developments in Language Theory* (Proceedings of DLT 2001, Vienna, Austria, July 16–21, 2001), LNCS 2295, 311–318.
10. E. L. Leiss, *Language equations*, Springer-Verlag, New York, 1999.
11. A. Okhotin, "Conjunctive grammars", *Journal of Automata, Languages and Combinatorics*, 6:4 (2001), 519–535.
12. A. Okhotin, "Conjunctive grammars and systems of language equations", *Programming and Computer Software*, 28:5 (2002), 243–249.
13. A. Okhotin, "Decision problems for language equations with Boolean operations", *Automata, Languages and Programming* (ICALP 2003, Eindhoven, The Netherlands, June 30–July 4, 2003), LNCS 2719, 239–251; journal version submitted.
14. A. Okhotin, "On the equivalence of linear conjunctive grammars to trellis automata", *Informatique Théorique et Applications*, 38 (2004), 69–88.
15. A. Okhotin, "On the number of nonterminals in linear conjunctive grammars", *Theoretical Computer Science*, 320:2–3 (2004), 419–448.
16. H. Rogers, Jr., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, 1967.
17. A. Salomaa, *Theory of Automata*, Pergamon Press, Oxford, 1969.
18. N. Yevtushenko, T. Villa, R. K. Brayton, A. Petrenko, A. L. Sangiovanni-Vincentelli, "Solution of parallel language equations for logic synthesis", *Proceedings of ICCAD 2001* (San Jose, CA, USA, November 4–8, 2001), 103–110.
19. G.-Q. Zhang, "Domain mu-calculus", *Informatique Théorique et Applications*, 37 (2003), 337–364.

Attacking the Common Algorithmic Problem by Recognizer P Systems

Mario J. Pérez Jiménez and Francisco J. Romero Campero[*]

Dpto. Computer Science and Artificial Intelligence
E.T.S. Ingeniería Informática. Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, España
{Mario.Perez, Francisco-Jose.Romero}@cs.us.es

Abstract. Many **NP**-complete problems can be viewed as special cases of the Common Algorithmic Problem (CAP). In a precise sense, which will be defined in the paper, one may say that CAP has a property of *local universality*. In this paper we present an effective solution to the decision version of the CAP using a family of recognizer P systems with active membranes. The analysis of the solution presented here will be done from the point of view of complexity classes in P systems.

Keywords: Membrane Computing, Common Algorithmic Problem, Cellular Complexity Classes.

1 Introduction

Membrane Computing is an emergent branch of Natural Computing. This unconventional model of computation is a kind of distributed parallel system, and it is inspired by some basic features of biological membranes.

Since Gh. Paun introduced it in [4], many different classes of such computing devices, called P systems, have already been investigated. Most of them are *computationally universal*, i.e., able to compute whatever a Turing machine can do, as well as *computationally efficient*, i.e., are able to trade space for time and solve in this way presumably intractable problems in a feasible time.

This paper deals with the Common Algorithmic Problem. This problem has the nice property (we can call this property *local universality*) that several other **NP**–complete problems can be reduced to it in linear time – we can say that they are *subproblems* of CAP. This property was already considered in [2], will be precisely defined in Section 2, and further illustrated in the paper.

Our study is focused on the design of a family of recognizer P systems solving it. We have followed the ideas and schemes used to solve other numerical **NP**-complete problems, such as Subsetsum in [6] and the Knapsack problem in [7].

[*] This work is supported by the Ministerio de Ciencia y Tecnología of Spain, by the *Plan Nacional de I+D+I (2000–2003)* (TIC2002-04220-C03-01), cofinanced by FEDER funds, and by a FPI fellowship (of the second author) from the University of Seville.

Due to the strong similarities of the design of these solutions the idea of a *cellular programming language* seems possible as it is suggested in [1].

The analysis of the presented solution will be done from the point of view of the complexity classes presented within the framework of the *complexity classes in P systems* studied in [5] and [9].

The paper is organized as follows. In the next section the Common Algorithmic Problem is presented as well as six **NP**-complete problems that can be viewed as "subproblems" of it. Section 3 recalls recognizer P systems with active membranes. In section 4 a polynomial complexity class for P systems is briefly introduced. Sections 5, 6 and 7 present a cellular solution to the Common Algorithmic Decision Problem. Conclusions are given in section 8.

2 The Common Algorithmic Problem

The *Common Algorithmic Problem* (CAP) [2] is the following: *let S be a finite set and F be a family of subsets of S. Find the cardinality of a maximal subset of S which does not include any set belonging to F. The sets in F are called forbidden sets.*

The Common Algorithmic Problem is an optimization problem, which can be transformed into a roughly equivalent decision problem by supplying a target value to the quantity to be optimized, and asking the question whether or not this value can be attained.

The Common Algorithmic Decision Problem (CADP) is the following: *Given S a finite set, F a family of subsets of S, and $k \in \mathbf{N}$, we ask if there exists a subset A of S such that $|A| \geq k$, and which does not include any set belonging to F.*

Definition 1. *We say that a problem X is a subproblem of another problem Y if there exists a linear–time reduction from X to Y (using a logarithmic bounded space).*

That is, X is a subproblem of Y if we can pass from problem X to problem Y through a simple rewriting process.

Next, we present some **NP**–complete problems that are subproblems of the CAP (or CADP).

2.1 The Maximum Independent Set Problem

The *Maximum Independent Set Problem* (MIS) is the following: *Given an undirected graph G, find the cardinality of a maximal independent subset I of G.*

The *Independent Set Decision Problem* (ISD) is the following: *Given an undirected graph G, and $k \in \mathbf{N}$, determine whether or not G has an independent set of size at least k.*

Theorem 1. *MIS* (*resp. ISD*) *is a subproblem of CAP* (*resp. CADP*).

2.2 The Minimum Vertex Cover Problem

The *Minimum Vertex Cover Problem* (MVC) is the following: *Given an undirected graph G, find the cardinality of a minimal set of a vertex cover of G.*

The *Vertex Cover Decision Problem* (VCD) is the following: *Given an undirected graph G, and $k \in \mathbf{N}$, determine whether or not G has a vertex cover of size at most k.*

Theorem 2. *MVC (resp. VCD) is a subproblem of CAP (resp. CADP).*

2.3 The Maximum Clique Problem

The *Maximum Clique Problem* (MAX-CLIQUE) is the following: *Given an undirected graph G, find the cardinality of a largest clique in G.*

The *Clique Decision Problem* is the following: *Given an undirected graph G, and $k \in \mathbf{N}$, determine whether or not G has a clique of size at least k.*

Theorem 3. *MAX-CLIQUE (resp. Clique Decision problem) is a subproblem of CAP (resp. CADP).*

2.4 The Satisfiability Problem

The *Satisfiability Problem* (SAT) is the following: *For a given set U of boolean variables and a finite set C of clauses over U, is there a satisfying truth assignment for C?*

Theorem 4. *Let $\varphi \equiv c_1 \wedge \cdots \wedge c_p$ be a boolean formula in conjunctive normal form. Let $Var(\varphi) = \{x_1, \ldots, x_n\}$, $c_i = l_{i,1} \vee \cdots \vee l_{i,r_i}$ $(1 \leq i \leq p)$, and $S = \{x_1, \ldots, x_n\} \cup \{\overline{x}_1, \ldots, \overline{x}_n\}$. For each i $(1 \leq i \leq p)$ let $A_i = \{\overline{l}_{i,1}, \ldots, \overline{l}_{i,r_i}\}$, considering $\overline{\overline{x}} = x$. Let $F = \{\{x_1, \overline{x}_1\}, \ldots, \{x_n, \overline{x}_n\}, A_1, \ldots, A_p\}$. Then the formula φ is satisfiable if and only if the solution of the CAP on input (S, F) is n.*

2.5 The Undirected Hamiltonian Path Problem

The *Undirected Hamiltonian Path Problem* is the following: *Given an undirected graph and two distinguished nodes u, v, determine whether or not there exists a path from u to v visiting each node exactly once.*

Theorem 5. *Let $G = (V, E)$ be an undirected graph, with $V = \{v_1, \ldots, v_n\}$. Then the following conditions are equivalent:*

(a) The graph G has a Hamiltonian path from v_1 to v_n.
(b) The solution of the CAP on input (S, F) is $n - 1$, where: $S = E$, and $F = \bigcup_{i=1}^{n} F_i$, with $F_i = \{B : B \subseteq B_i \ |B| = 2\}$, for $i = 1, n$, and $F_i = \{B : B \subseteq B_i \ \wedge |B| = 3\}$, for all $1 < i < n$, with $B_i = \{\{v_i, u\} : \{v_i, u\} \in E\}$.

2.6 The Tripartite Matching Problem

The *Tripartite Matching Problem* is the following: *Given three sets B, G, and H, each containing n elements, and a ternary relation $T \subseteq B \times G \times H$, determine whether or not there exists a subset T' of T such that $|T'| = n$ and no two of triples belonging to T' have a component in common.*

We say that T' is a tripartite matching associated with (B, G, H, T).

Theorem 6. *Let $B = \{b_1, \ldots, b_n\}$, $G = \{g_1, \ldots, g_n\}$, and $H = \{h_1, \ldots, h_n\}$, be sets containing n elements. Let T be a subset of $B \times G \times H$. Then the following conditions are equivalent:*

(a) There exists a tripartite matching associated with (B, G, H, T).
(b) The solution of the CAP on input (S, F) is n, where $S = T$ and $F = \bigcup_{i=1}^{n}(F_{b_i} \cup F_{g_i} \cup F_{h_i})$, with $F_{b_i} = \{A : \ |A| = 2 \ \wedge \ A \subseteq \{(b_i, g, h) : \ (b_i, g, h) \in T\}\}$, $F_{g_i} = \{A : \ |A| = 2 \ \wedge \ A \subseteq \{(b, g_i, h) : \ (b, g_i, h) \in T\}\}$, and $F_{h_i} = \{A : \ |A| = 2 \ \wedge \ A \subseteq \{(b, g, h_i) : \ (b, g, h_i) \in T\}\}$, for all $1 \leq i \leq n$.

3 Recognizer P Systems with Active Membranes

Definition 2. *A* P system with input *is a tuple (Π, Σ, i_Π), where: (a) Π is a P system, with working alphabet Γ, with p membranes labelled by $1, \ldots, p$, and initial multisets $\mathcal{M}_1, \ldots, \mathcal{M}_p$ associated with them; (b) Σ is an (input) alphabet strictly contained in Γ; the initial multisets are over $\Gamma - \Sigma$; and (c) i_Π is the label of a distinguished (input) membrane.*

Let m be a multiset over Σ. The *initial configuration of (Π, Σ, i_Π) with input m* is $(\mu, \mathcal{M}_1, \ldots, \mathcal{M}_{i_\Pi} \cup m, \ldots \mathcal{M}_p)$.

Definition 3. *Let $\mu = (V(\mu), E(\mu))$ be a membrane structure. The* membrane structure with environment *associated with μ is the rooted tree such that: (a) the root of the tree is a new node that we will denote env; (b) the set of nodes is $V(\mu) \cup \{env\}$; and (c) the set of edges is $E(\mu) \cup \{\{env, skin\}\}$. The node env is called the* environment *of the structure μ.*

In the case of P systems with input and with external output, the concept of computation is introduced in a similar way as for standard P systems – see [3]– but with a slight change. Now the configurations consist of a membrane structure with environment, and a family of multisets of objects associated with each region and with the environment.

Next we introduce P systems able to accept or reject multisets considered as inputs. Therefore, these systems will be suitable to attack the solvability of decision problems.

Definition 4. *A recognizer P system is a P system with input (Π, Σ, i_Π), and with external output such that: (a) the working alphabet contains two distinguished elements YES, NO; (b) all its computations halt; and (c) if $\mathcal{C}$ is a computation of Π, then either the object YES or the object NO (but not both) have to be sent out to the environment, and only in the last step of the computation.*

Definition 5. *We say that $\mathcal{C}$ is an accepting computation (respectively, rejecting computation) if the object YES (respectively, NO) appears in the environment associated with the corresponding halting configuration of $\mathcal{C}$.*

These recognizer P systems are specially adequate when we are trying to solve a decision problem. In this paper we will deal with P systems with active membranes. We refer to [3] (see chapter 7, section 7.2) for a detailed definition of evolution rules, transition steps, configurations and computations in this model.

Let us denote by $\mathcal{AM}$ the class of recognizer P systems with active membranes using 2-division (for elementary membranes).

4 The Complexity Class $\mathbf{PMC}_{\mathcal{F}}$

Roughly speaking, a computational complexity study of a solution to a problem is an estimation of the resources (time, space, etc) that are required through all processes that take place in the way from the bare instance of the problem up to the final answer.

The first results about "solvability" of **NP**–complete problems in polynomial time (even linear) by cellular computing systems with membranes were obtained using variants of P systems that lack an input membrane. Thus, the constructive proofs of such results need to design one system for each instance of the problem.

This drawback can be easily avoided if we consider P systems with input. Then, the same system could solve different instances of the problem, provided that the corresponding input multisets are introduced in the input membrane.

Instead of looking for a single system that solves a problem, we prefer designing a family of P systems such that each element of the family decides all the instances of "equivalent size" of the problem.

Let us now introduce some basic concepts before the definition of the complexity class itself.

Definition 6. *Let L be a language, and $\mathbf{\Pi} = (\Pi(n))_{n\in\mathbf{N}}$ be a family of P systems with input. A polynomial encoding of L in $\mathbf{\Pi}$ is a pair (g, h) of polynomial-time computable functions $g : L \to \bigcup_{n\in\mathbf{N}} I_{\Pi(n)}$ and $h : L \to \mathbf{N}$, such that for every $u \in L$ we have $g(u) \in I_{\Pi(h(u))}$.*

That is, for each string $u \in L$, we have a multiset $g(u)$ and a number $h(u)$ associated with it such that $g(u)$ is an input multiset for the P system $\Pi(h(u))$.

Lemma 1. *Let $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ be languages. Let $\mathbf{\Pi} = (\Pi(n))_{n\in\mathbf{N}}$ a family of P systems with input. If $r : \Sigma_1^* \to \Sigma_2^*$ is a polynomial time reduction from L_1 to L_2, and (g, h) is a polynomial encoding of L_2 in $\mathbf{\Pi}$, then $(g \circ r, h \circ r)$ is a polynomial encoding of L_1 in $\mathbf{\Pi}$.*

For a detailed proof, see [9].

Definition 7. *Let $\mathcal{F}$ be a class of recognizer P systems. A decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family of P systems $\mathbf{\Pi} = (\Pi(n))_{n\in\mathbf{N}}$ from $\mathcal{F}$, and we denote this by $X \in \mathbf{PMC}_{\mathcal{F}}$, if the following is true:*

- *The family* $\mathbf{\Pi}$ *is consistent with regard to the class* $\mathcal{F}$*; that is,* $\forall t \in \mathbf{N}$ $(\Pi(t) \in \mathcal{F})$.
- *The family* $\mathbf{\Pi}$ *is polynomially uniform by Turing machines; that is, there exists a deterministic Turing machine constructing* $\Pi(t)$ *from* t *in polynomial time.*
- *There exists a polynomial encoding* (g, h) *from* I_X *to* $\mathbf{\Pi}$ *such that:*
  - *The family* $\mathbf{\Pi}$ *is polynomially bounded with regard to* (X, g, h)*; that is, there exists a polynomial function* p*, such that for each* $u \in I_X$ *every computation of* $\Pi(h(u))$ *with input* $g(u)$ *is halting and, moreover, it performs at most* $p(|u|)$ *steps.*
  - *The family* $\mathbf{\Pi}$ *is sound with regard to* (X, g, h)*; that is, for each* $u \in I_X$*, if there exists an accepting computation of* $\Pi(h(u))$ *with input* $g(u)$*, then* $\theta_X(u) = 1$.
  - *The family* $\mathbf{\Pi}$ *is complete with regard to* (X, g, h)*; that is, for each* $u \in I_X$*, if* $\theta_X(u) = 1$*, then every computation of* $\Pi(h(u))$ *with input* $g(u)$ *is an accepting one.*

In the above definition we have imposed to every P system $\Pi(n)$ to be *confluent*, in the following sense: every computation with the *same* input produces the *same* output; that is, for every input multiset m, either every computation of $\Pi(n)$ with input m is an accepting computation, or every computation of $\Pi(n)$ with input m is a rejecting computation.

Proposition 1. *Let* $\mathcal{F}$ *be a class of recognizer P systems. Let* X, Y *be problems such that* X *is reducible to* Y *in polynomial time. If* $Y \in \mathbf{PMC}_{\mathcal{F}}$*, then* $X \in \mathbf{PMC}_{\mathcal{F}}$.

For a detailed proof, see [9].

5 Solving CADP by Recognizer P Systems

We will address the resolution of this problem via a brute force algorithm, in the framework of recognizer P systems with active membranes using 2-division, and without cooperation nor priority among rules. Our strategy will consist in the following phases:

- *Generation stage:*
  1. At the beginning there will be only one internal membrane which will represent the set $A = S$.
  2. For each subset $B \in F$ do:

```
       if $B \subseteq A$ then
         for each $e \in B$ do:
           Using membrane division generate one membrane
           representing the subset $A - \{ e \}$
       end if
```

- *Calculation stage:* In this stage the system calculates the cardinality of the subset associated with each membrane.

– *Checking stage:* Here the system checks whether or not the cardinality of each generated subset exceeds the goal k.
– *Output stage:* According to the previous stage, one object YES or one object NO is sent out to the environment.

Now we construct a family of P systems with active membranes using 2-division solving the *Common Algorithmic Decision Problem.*
Let us consider a polynomial bijection, $\langle\,\rangle$, between $\mathbb{N}^3$ and $\mathbb{N}$ (e.g., $\langle\, x,\, y,\, z\,\rangle = \langle\,\langle\, x,\, y\,\rangle,\, z\,\rangle$) induced by the pair function $\langle x, y\rangle = (x+y)\cdot(x+y+1)/2+x$).

The family of P systems with input considered here is

$$\mathbf{\Pi} = \{\ (\Pi(\langle n,m,k\rangle), \Sigma(n,m,k), i(n,m,k))\ :\ (n,m,k)\in\mathbb{N}^3\ \}$$

For each $(n,m,k)\in\mathbb{N}^3$, we have $\Sigma(n,m,k) = \{\, s_{ij}\ :\ 1\le i\le m,\ 1\le j\le n\}$, $i(n,m,k)=2$, and $\Pi(\langle n,m,k\rangle) = (\Gamma(n,m,k), \{1,2\}, \mu, \mathcal{M}_1, \mathcal{M}_2, R)$ is defined as follows:

– Working alphabet:

$$\begin{aligned}\Gamma(n,m,k) = \Sigma(n,m,k) &\cup \{a_i\ :\ 1\le i\le m\ \} \cup \{c_i\ :\ 0\le i\le 2n+1\ \}\\ &\cup \{ch_i\ :\ 0\le i\le 2k+1\ \} \cup \{f_j\ :\ 1\le j\le n+1\ \}\\ &\cup \{e_{i,\,j,\,l}\ :\ 1\le i\le m, 1\le j\le n, -1\le l\le j+1\ \}\\ &\cup \{g_j\ :\ 0\le j\le nm+m+1\ \}\\ &\cup \{z,\ s_+,\ s_-,\ S_+,\ S_-,\ S,\ o,\ \tilde{O},\ O,\ p,\ t,\ neg,\ i_1,\ i_2\}\\ &\cup \{YES_0,\ YES_1,\ YES_2,\ YES,\ preNO,\ NO\ \}.\end{aligned}$$

– Membrane structure: $\mu = [\ [\ \ ]_2\ ]_1$ (we will say that every membrane with label 2 is an *internal membrane*).
– Initial multisets: $\mathcal{M}_1 = \emptyset$; $\mathcal{M}_2 = \{g_0, z^m, s_+^n, o^k\}$.
– The set of evolution rules, R, consists of the following rules:
(1) $[\ s_{1,\,j}\ \rightarrow\ f_j\ ]_2^0$, for $1\le j\le n$,
$[\ s_{i,\,j}\ \rightarrow\ e_{i,\,j,\,j}\ ]_2^0$, for $2\le i\le m,\ 1\le j\le n$.
The objects $s_{i,\,j}$ encode in the initial configuration the *forbidden sets.* The presence of an object $s_{i,\,j}$ indicates that $s_j\in B_i$. The objects of type f represent the elements of the *forbidden set* that is being analized and the objects e represent the rest of the *forbidden sets.*
(2) $[\ f_1\ ]_2^0\ \rightarrow\ [\ \sharp\ ]_2^0\ [\ s_-\ ]_2^+$.
The goal of these rules is to generate membranes for subsets A of S such that $\forall\, B\in F\,(\,B\not\subseteq A\,)$. The system, in order to ensure the condition $B\not\subseteq A$, eliminates from A one element of the *forbidden set* B.
(3) $[\ f_{j'}\ \rightarrow\ f_{j'-1}\ ]_2^0$, for $2\le j'\le n+1$,
$[\ e_{i,\,j,\,l}\ \rightarrow\ e_{i,\,j,\,l-1}\ ]_2^0$, for $2\le i\le m,\ 1\le j\le n,\ 0\le l\le j+1$.
During the computation for a *forbidden subset*, B, the above rules perform a rotation of the subscript of the objects f and of the third subscript of the objects e. These subscripts represent the order in which the elements are considered to be eliminated from the subset A associated with the membrane in order to ensure the condition $B\not\subseteq A$.
(4) $[\ f_{j'}\ \rightarrow\ \sharp\ ]_2^+$, for $1\le j'\le n+1$,
$[\ e_{i,\,j,\,0}\ \rightarrow a_{i-1}\ ]_2^+$, for $2\le i\le m,\ 1\le j\le n$.

When the polarization of an internal membrane is positive during the *generation stage* the associated subset A fulfills the condition $B \nsubseteq A$, where B is the *forbidden set* that is being analized. In this situation the elements of the current *forbidden set* are *erased* by these rules. Moreover, if the element removed from A is a member of the *forbidden set* B_i, then the object a_{i-1} appears in the membrane to certify that the associated subset also fulfills the condition $B_i \nsubseteq A$.

(5) $[\, e_{i,j,-1} \ \rightarrow \ e_{i-1,j,j+1} \,]_2^+$, for $3 \le i \le m$, $1 \le j \le n$,
$[\, e_{i,j,l} \ \rightarrow \ e_{i-1,j,j+1} \,]_2^+$, for $3 \le i \le m$, $1 \le j \le n$, $1 \le l \le j+1$,
$[\, e_{2,j,l} \ \rightarrow \ f_{j+1} \,]_2^+$, for $1 \le j \le n$, $1 \le l \le j+1$,
$[\, e_{2,j,-1} \ \rightarrow \ f_{j+1} \,]_2^+$, for $1 \le j \le n$,
$[\, a_{i'} \ \rightarrow \ a_{i'-1} \,]_2^+$, for $2 \le i' \le m$.

In order to continue the computation for the next *forbidden subset* B, these rules perform a rotation of the subscripts of the objects e and a. Note that the subscript representing the position of the element to be analized is set one position ahead in order to allow the system to check whether the current associated subset A satisfies the condition $B \nsubseteq A$.

(6) $[\, a_1 \,]_2^0 \ \rightarrow \ \sharp\, [\ ]_2^+$; $[\, a_1 \ \rightarrow \ \sharp \,]_2^+$.

The presence of object a_1 in a neutrally charged internal membrane means that the *forbidden set* which is going to be analized already satisfies the condition $B \nsubseteq A$; consequently this object changes the polarization of the membrane to positive in order to skip the computation for this *forbidden subset.*

(7) $[\, z \,]_2^+ \ \rightarrow \ \sharp\, [\ ]_2^0$.

The object z sets the polarization of the internal membranes to neutral once the generation stage for one forbidden set has taken place.

(8) $[\, g_j \ \rightarrow \ g_{j+1} \,]_2^0$, $[\, g_j \ \rightarrow \ g_{j+1} \,]_2^+$, for $0 \le j \le nm+m$,
$[\, g_{nm+m+1} \ \rightarrow \ neg,\, c_0 \,]_2^0$.

The objects g are counters used in the *generation stage.*

(9) $[\, neg \,]_2^0 \ \rightarrow \ \sharp\, [\ ]_2^-$, $[\, z \,]_2^- \ \rightarrow \ \sharp$.

The multiplicity of the object z represents the number of *forbidden sets* that do not satisfy the condition $B \nsubseteq A$. So, if there is an object z in an internal membrane when the *generation stage* is over, then the membrane is dissolved.

(10) $[\, s_+ \ \rightarrow \ S_+ \,]_2^-$, $[\, s_- \ \rightarrow \ S_- \,]_2^-$, $[\, o \ \rightarrow \ \tilde{O} \,]_2^-$.

The multiplicity of the object s_+ represents the cardinality of S, the multiplicity of the object s_- represents the number of removed elements from S and the multiplicity of the object o represents the constant k. At the begining of the *calculation stage* the objects s_+, s_-, and o are renamed to S_+, S_-, and $\tilde{O}$ in order to avoid the interference with the previous stage.

(11) $[\, S_- \,]_2^- \ \rightarrow \ \sharp\, [\ ]_2^+$, $[\, S_+ \,]_2^+ \ \rightarrow \ \sharp [\ ]_2^-$.

These rules are used to calculate the cardinality of the subsets associated with the internal membranes.

(12) $[\, c_i \ \rightarrow \ c_{i+1} \,]_2^-$, $[\, c_i \ \rightarrow \ c_{i+1} \,]_2^+$, for $0 \le i \le 2n$,
$[c_{2n+1} \ \rightarrow \ t,\, ch_0]_2^-$.

The objects c are counters used in the *calculation stage.*

(13) $[\, t \,]_2^- \ \rightarrow \ \sharp\, [\ ]_2^0$, $[\, S_+ \ \rightarrow \ S \,]_2^0$, $[\, \tilde{O} \ \rightarrow \ O \,]_2^0$.

The object t will change the polarization of the internal membranes to neutral starting the *checking stage*. In this stage the objects S_+ and $\tilde{O}$ are renamed to S and O, in order to avoid the interference with the previous stages.

(14) $[\, S \,]_2^0 \rightarrow \sharp\, [\ ]_2^+, [\, O \,]_2^+ \rightarrow \sharp\, [\ ]_2^0$

These rules are used to compare the multiplicity of the objects S and O.

(15) $[\, ch_i \rightarrow ch_{i+1} \,]_2^0, [\, ch_i \rightarrow ch_{i+1} \,]_2^+, \quad 0 \leq i \leq 2k.$

The objects ch are counters in the *checking stage*.

(16) $[\, ch_{2k+1} \,]_2^+ \rightarrow YES_0\, [\ ]_2^-, [\, ch_{2k+1} \rightarrow p, i_1 \,]_2^0.$

The *checking stage* finishes when the object ch_{2k+1} appears in the internal membranes. These rules are used to send the object YES_0 to the skin, if the charge is positive, and to check whether the answer must be *NO*.

(17) $[\, p \,]_2^0 \rightarrow \sharp\, [\ ]_2^+, [\, i_1 \rightarrow i_2 \,]_2^+,$
$[\, i_2 \,]_2^+ \rightarrow YES\, [\ ]_2^-, [\, i_2 \,]_2^0 \rightarrow preNO\, [\ ]_2^-.$

These rules decide if there are any objects O in the internal membranes when the *checking stage* is over, in order to send out the right answer.

(18) $[\, YES_i \rightarrow YES_{i+1} \,]_1^0$, for $0 \leq i \leq 1$,
$[\, YES_2 \rightarrow YES \,]_1^0, [\, preNO \rightarrow NO \,]_1^0.$

These rules are used to sinchronize the *output stage*.

(19) $[\, YES \,]_1^0 \rightarrow YES\, [\ ]_1^+, [\, NO \,]_1^0 \rightarrow NO\, [\ ]_1^-.$

These rules send out the answer to the environment.

6 An Overview of the Computation

First of all we define a polynomial encoding for the *CADP* in $\mathbf{\Pi}$ in order to study its computational complexity. Let $h(u) = \langle n, m, k \rangle$ and $g(u) = \{ s_{i,j} : s_j \in B_i \}$, for a given *CADP*–instance $u = (\{ s_1, \ldots, s_n \}, (B_1, \ldots, B_m), k)$. Next we informally describe how the system $\Pi(h(u))$ with input $g(u)$ works.

In the first step of the computation, according to the rules in (1), the objects s evolve to the objects f and e. The objects f represent the elements of the *forbidden set* that is being analized and the objects e represent the others.

The generation stage takes place following the rules from group (2) - (8). The systems generates subsets of S with the greatest possible cardinalities and associates them with internal membranes. Let us describe the evolution of the *subsets associated with internal membranes* during the *generation stage.*

The *subset associated* with the initial internal membrane is $A = S$.

When the object f_1 appears in a neutrally charged internal membrane, during the *generation stage* for a *forbidden set* B, using the rule in (2) the system produces two new membranes: one (positively charged) where the analized element is removed, and another one (neutrally charged) where an element of B (different from f_1) is removed in order to achieve the condition $B \not\subseteq A$. These two new membranes behave in a different way.

On the one hand in order to study the next element, in the neutrally charged membrane the rules in (3) perform a rotation of the subscript of the objects f and of the third subscript of the objects e, which represent the order in which the elements are considered.

On the other hand, in the positively charged membrane an object s_- appears, indicating that one element has been removed from the associated subset A. The system moves on to analize the remaining *forbidden sets* rotating the first subscripts of the objects e according to the rules in (4) and (5), and *erasing* the objects f. Note that the third subscript of the objects e and the subscript of the object f are set one position ahead in order to check whether the condition imposed by the *forbidden set* is satisfied. The computation corresponding to a *forbidden set* finishes when an object z changes the polarization from positive to neutral following the rule in (7).

In the first step of the computation for a *forbidden set* the position of the elements in which they will be studied are one position ahead and so the system has a step to check whether there exists an object a_1 which means that the associated subset already fulfills the condition $B \nsubseteq A$. This is done applying the rules in (6). The objects a appear by applying the second rule in (4) when the system removes an element belonging to several *forbidden sets*.

The *generation stage* ends when the object g_{nm+m+1} appears. Between the *generation stage* and the *calculation stage* there is a gap of two steps of transition. In the first step, according to the last rule in (8), the object g_{nm+m+1} evolves to the object c_0 (a counter for the *calculation stage*) and the object *neg*. This object changes the polarization of the internal membranes to negative using the first rule in (9). In the second step, one dissolves the membranes whose associated subsets A have the property that there exists $B \in F$ such that $B \subseteq A$. The number of sets in F verifying $B \subseteq A$ is represented by the multiplicity of the object z. So, at the end of the *generation stage*, when the polarization is negative, if there is an object z, then the membrane is dissolved by the second rule in (9). Moreover, in this step the objects s_+, s_-, and o are renamed to S_+, S_-, and $\tilde{O}$ by the rules in (10), in order to avoid interference with the previous stage.

The multiplicity of the object S_+ encodes the cardinality of the set S and the multiplicity of the object S_- encodes the number of elements that have been removed from S to construct the final associated subset A. Thus, in order to calculate the cardinality of A the system applies the rules in (11), which implement the subtraction $\mathit{multiplicity}(S_+) - \mathit{multiplicity}(S_-)$ in each internal membrane.

The *calculation stage* ends when the object c_{2n+1} evolves to the object ch_0 (a counter in the *checking stage*) and the object t. By using the rule in (13), t changes the polarization of the internal membranes from negative to neutral.

In the transition stage from the *calculation stage* to the *checking stage* the objects S_+ and $\tilde{O}$ are renamed to S and O using the rules in (13) in order to avoid interference with the previous stages.

In the *checking stage*, by using the rules in (14), the system decides in each internal membrane if the multiplicity of the object S, encoding the cardinality of the *associated subset*, is greater than or equal to the multiplicity of the object O, encoding the constant k.

The *checking stage* ends when the object ch_{2k+1} appears in the internal membrane and then the *output stage* starts. If the object ch_{2k+1} appears when the membrane is positively charged, then the number of objects S exceeded the number of object O and so, by using the first rule in (16), the object YES_0 is sent to the skin region. This object has to evolve to YES_1, YES_2 and, finally, to YES in order to synchronize the *output stage*. On the other hand, if the object ch_{2k+1} appears in a neutrally charged membrane, then the number of objects S was less than or equal to the number of objects O. In this situation the object ch_{2k+1}, following the second rule in (16), evolves to i_1 and p. This object changes the polarization of the internal membranes to positive in order to allow any remaining objects O to set it again to neutral according to the rules in (14). While the membrane is positively charged the object i_1 evolves to i_2. If i_2 appears in a positively charged internal membrane, then there were no objects O, therefore the multiplicity of the object S was equal to the multiplicity of the object O, and so the object YES is sent to the skin region according to the rules in (17). On the other hand, if the object i_2 appears in a neutrally charged membrane, then there were objects O and so the object $preNO$ is sent to the skin region.

In the last step of the computation the rules in (19) send out the answer to the environment. Note that the occurrence of the objects NO is delayed one step, by the rule $[\, preNO \rightarrow NO]_1^0$, in order to allow the system to send out the object YES, if any.

7 Required Resources

The presented family of recognizer P systems solving the *Common Algorithmic Decision Problem* is polynomially uniform by Turing machines. Note that the definition of the family is done in a recursive manner from a given instance, in particular, from the constants n, m, and k. Futhermore, the resources required to build an element of the family are the following:

- Size of the alphabet: $n^2m + 4nm + 2n + 3m + 2k + 24 \in O((max\{n, m, k\})^3)$.
- Number of membranes: $2 \in \Theta(1)$.
- $|\mathcal{M}_1| + |\mathcal{M}_2| = n + m + k + 1 \in O(n + m + k)$.
- Sum of the rules' lengths: $12n^2m + 12n^2 + 49nm + 71n + 25m + 30k + 242 \in O((max\{n, m, k\})^3)$.

The instance $u = (\{\, s_1, \ldots, s_n \,\}, (\{\, s_1^1, \ldots, s_{r_1}^1 \,\}, \ldots, \{\, s_1^m, \ldots, s_{r_m}^m \,\}), k)$ is introduced in the initial configuration through an input multiset; that is, it is encoded in an *unary representation* and, thus, we have that $|u| \in O(n + m + k)$.

The number of steps in each stage are the following:

1. *Generation stage:* $nm + m + 1$ steps.
2. *Transition to the calculation stage:* 2 steps.
3. *Calculation stage:* $2n + 1$ steps.
4. *Transition to the checking stage:* 2 steps.
5. *Checking stage:* $2k + 2$ steps.
6. *Output stage:* 6 steps.

So, the overall number of steps is $nm + 2n + m + 2k + 14 \in O(max\{n, m, k\}^2)$.

From these discussions we deduce the following results:

Theorem 7. *$CADP \in \mathbf{PMC}_{\mathcal{AM}}$.*

Corollary 1. *$\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{AM}}$, and $\mathbf{NP} \cup \mathbf{co-NP} \subseteq \mathbf{PMC}_{\mathcal{AM}}$.*

8 Conclusions

Many **NP**-complete problems can be viewed as special cases of an optimization problem called Common Algoritmic Problem. In this work the importance of this problem is emphasized by the presentation of six relevant **NP**-complete problems that are "subproblems" of it (or of its corresponding decision version). Furthermore, a solution to the Common Algoritmic Decision Problem by a family of recognizer P systems with active membranes is presented.

The study and design of solutions to *locally universal* problems as CAP and CADP within the framework of unconventional computing models like P systems seems very interesting because these solutions may give, in some sense, *patterns* that can be used for attacking the solvability of many **NP**-complete problems.

References

1. Gutiérrez–Naranjo, M.A.; Pérez–Jiménez, M.J.; Riscos–Núñez, A. Towards a programming language in cellular computing. In: Gh. Păun; A. Riscos–Núñez, A. Romero–Jiménez; F. Sancho–Caparrini (eds.) *Proceedings of the Second Brainstorming Week on Membrane Computing*, Report RGNC 01/04, University of Seville, Spain, 2004, 247–257.
2. Head, T.; Yamamura, M.; Gal, S. Aqueous computing: writing on molecules. *Proceedings of the Congress on Evolutionary Computation 1999*, IEEE Service Center, Piscataway, NJ, 1999, 1006–1010.
3. Păun, Gh.: *Membrane Computing. An Introduction*, Springer-Verlag, 2002
4. Păun, Gh.: Computing with membranes. *Journal of Computer and Systems Sciences*, 61(1), 2000, 108–143.
5. Pérez–Jiménez, M.J.; Romero–Jiménez, A.; Sancho–Caparrini, F.: *Teoría de la Complejidad en modelos de computacion celular con membranas*, Ed. Kronos, 2002.
6. Pérez-Jiménez, M.J.; Riscos-Núñez, A. Solving the Subset-Sum problem by active membranes, submitted.
7. Pérez–Jiménez, M.J.; Riscos–Núñez, A. A linear-time solution for the Knapsack problem using active membranes. *Lecture Notes in Computer Science*, 2933 (2004) 140–152.
8. Pérez–Jiménez, M.J.; Romero–Campero, F.J. A CLIPS simulator for recognizer P systems with active membranes. In: Gh. Păun; A. Riscos–Núñez; A. Romero–Jiménez; F. Sancho–Caparrioni (eds.) *Proceedings of the Second Brainstorming Week on Membrane Computing*, Report RGNC 01/04, University of Seville, Spain, 2004, 387–413.
9. Pérez–Jiménez, M.J.; Romero–Jiménez, A.; Sancho–Caparrini, F. A polynomial complexity class in P systems using membrane division. In: E. Csuhaj-Varjú; C. Kintala; D. Wotschke; Gy. Vaszil (eds.) *Proceedings of the Fifth International Workshop on Descriptional Complexity of Formal Systems*, 2003, 284–294.

On the Minimal Automaton of the Shuffle of Words and Araucarias

Extended Abstract

René Schott[1] and Jean-Claude Spehner[2]

[1] LORIA and IECN, Université Henri Poincaré, 54506 Vandoeuvre-lès-Nancy, France
schott@loria.fr

[2] Laboratoire MAGE, FST, Université de Haute Alsace, 68093, Mulhouse, France
JC.Spehner@uha.fr

Abstract. The shuffle of k words $u_1, \ldots, u_k$ is the set of words obtained by interleaving the letters of these words such that the order of appearance of all letters of each word is respected. The study of the shuffle product of words leads to the construction of an automaton whose structure is deeply connected to a family of trees which we call araucarias. We prove many structural properties of this family of trees and give some combinatorial results. The link with the minimal partial automaton which recognizes the words of the shuffle is established. Our method works for the shuffle of $k \geq 2$ words.

Keywords: Automaton, shuffle of words, trees.

1 Introduction

Partial commutations have been intensively investigated over the two last decades in connection with parallel processing [3,4,6,8,15]. Among the available publications let us mention the works done by the present authors how have designed an efficient sequential algorithm for the generation of commutation classes [12] and two optimal parallel algorithms on the commutation class of a given word [13]. During our investigation, it became clear to us that some of our results extend to the shuffle product of words. Works on shuffle products are sparse (see [1,2,7,9,10,11,14]) and many problems remain open. It has been recognized that these problems are difficult both from algebraic, combinatorial and algorithmic point of views. [14] gives an algorithm which determines the shuffle of two words. Partial commutation theory assumes that a letter never commutes with itself but, in the shufle product, the occurrences of the same letter of two distinct words still commute. This fundamental difference implies that the minimal automaton of the shuffle of words contains subautomata whose graph is the opposite of a directed tree. This paper is a (small) step towards the solution of open problems related to the shuffle product of $k \geq 2$ words written on alphabets which are not disjoint. More precisely, it is devoted to the study of the above mentioned

subautomata and, in particular to the related directed trees which we call araucarias.
In Section 2 we give a direct definition of these araucarias, characterize their maximal paths and finally give a construction based on the properties of the terminal sections of the maximal paths. Then we define a family of remarkable symmetric polynomials which play a crucial role in the computation of the size of the araucarias.
Section 3 is devoted to the study of the minimal automaton of the shuffle product of words. We prove that, under some assumptions, this automaton contains the opposite of an araucaria.
Do to the lack of space some proofs are omitted.

2 Araucarias

2.1 Basic Definitions and Properties

Below we give a direct definition which is independent of the automata which recognize the shuffle of words.

Definition 1. *Any path of length p is called araucaria of type (p) and arity 1. Such an araucaria is said to be elementary.*
Let k be a strictly positive integer and $(p_1, \ldots, p_k)$ a sequence of k strictly positive integers. Assume that the araucarias of arity $k-1$ have been defined.
Any directed tree A such that:
- A admits a unique path τ of length $p = p_1 + \ldots + p_k$ called trunk of A;
- for each $i \in \{1, \ldots, k\}$ and for each $h \in \{0, \ldots, p_i\}$, A admits a subtree $A_{i,h}$ which is isomorphic to an araucaria of type $(p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_k)$ whose root is the node s_h of τ of height h and whose leafs are leafs of A;
- for each $i \in \{1, \ldots, k\}$, the subtrees $A_{i,0}, \ldots, A_{i,p_i}$ are two by two disjoint;
- for all $i, j \in \{1, \ldots, k\}$ $(i \neq j)$ and h such that $0 \leq h \leq min(p_i, p_j)$ and $h \neq max(p_i, p_j)$, the trunks of the araucarias $A_{i,h}$ and $A_{j,h}$ have only their root s_h in common;
- A is of minimal size
is called araucaria of type $(p_1, \ldots, p_k)$ and arity k. Such an araucaria is unique up to an isomorphism by the proof of Theorem 1 (see below) and is denoted $A(p_1, \ldots, p_k)$.
In this definition the minimality condition imposes that:
- for all $i, j \in \{1, \ldots, k\}$ $(i \neq j)$ the trunks of the araucarias A_{i,p_i} and A_{j,p_j} have a common terminal section of length $min(p_i, p_j)$ which is contained in τ;
- for all $i, j \in \{1, \ldots, k\}$ $(i < j)$ and $h \in \{0, \ldots, min(p_i, p_j)\}$, such that $h \neq max(p_i, p_j)$, $A_{i,h}$ and $A_{j,h}$ admit a common subaraucaria of type $(p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_{j-1}, p_{j+1}, \ldots, p_k)$ (see Figures 1 and 2).
Each path issued from the root of a directed tree A and whose last node is a leaf of A is said to be maximal.

Definition 1 is by induction and allows to see that, in an araucaria, each maximal path starts with a section which is linked to an elementary subaraucaria. The following lemma is deduced from that by induction.

Lemma 1. *If A is an araucaria of type $(p_1, \ldots, p_k)$, for each maximal path σ of A, there exist a factorisation $\sigma_1 \ldots \sigma_h$ of σ with $1 \leq h \leq k$ and a one-to-one mapping η from $\{\sigma_1, \ldots, \sigma_h\}$ into $\{1, \ldots, k\}$ such that $\forall i \in \{1, \ldots, h-1\}$, $0 < |\sigma_i| \leq p_{\eta(\sigma_i)}$ and $|\sigma_h| = p_{\eta(\sigma_h)}$*

Proof. The property is trivial for $k = 1$. Assume that the property is verified for each araucaria of arity smaller than or equal to $k-1$ ($k > 1$) and let A be an araucaria of arity k and of type $(p_1, \ldots, p_k)$ and let $\sigma = (s_0, s_1, \ldots, s_f)$ be a maximal path of A.
By Definition 1, there exists a node s_q of A's trunk which is distinct from s_0 and a subaraucaria A' of A with arity less than or equal to $k-1$ and with root s_q which contains the section $\sigma' = (s_q, \ldots, s_f)$ of σ. Then there exists $i_1 \in \{1, \ldots, k\}$ such that the type of A' is equal to $(p_1, \ldots, p_{i_1-1}, p_{i_1+1}, \ldots, p_k)$ or to one of its subsequences and the section $\sigma_1 = (s_0, \ldots, s_q)$ can be linked to the elementary araucaria $A(p_{i_1})$. If we set $\eta(\sigma_1) = i_1$, we have $0 < |\sigma_1| \leq p_{i_1} = p_{\eta(\sigma_1)}$.
By the induction hypothesis, σ' admits a factorisation into at most $k-1$ sections $\sigma_2, \ldots, \sigma_h$ and there exists a one-to-one mapping η' from $\{\sigma_2, \ldots, \sigma_h\}$ to $\{1, \ldots, k\} \setminus \{i_1\}$ such that $\forall i \in \{2, \ldots, h-1\}$, $0 < |\sigma_i| \leq p_{\eta(\sigma_i)}$ and $|\sigma_h| = p_{\eta(\sigma_h)}$.
If we set $\eta(\sigma_i) = \eta'(\sigma_i)$ for all $i \in \{2, \ldots, h\}$, then η has the properties required by the lemma. $\square$

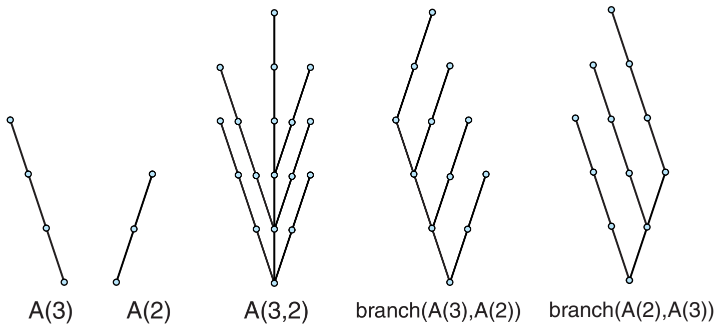


Fig. 1. Araucarias of arity 1 and 2 and ramified directed trees.

Definition 2. *Let $\{A(p_1), \ldots, A(p_k)\}$ be a set of elementary araucarias, A a directed tree, σ a maximal path of A and $\sigma_1 \ldots \sigma_h$ a factorisation of σ in sections with strictly positive lengths.*
(i) Every one-to-one mapping η from the set $\{\sigma_1, \ldots, \sigma_h\}$ into $\{1, \ldots, k\}$ such that, $\forall i \in \{1, \ldots, h-1\}$, $|\sigma_i| \leq p_{\eta(\sigma_i)}$ and $|\sigma_h| = p_{\eta(\sigma_h)}$ is called an attribution function to $\{A(p_1), \ldots, A(p_k)\}$.
For each $i \in \{1, \ldots, h\}$, the section σ_i is said to be attributable to the elementary araucaria $A(p_{\eta(\sigma_i)})$.
If $|\sigma_i| = p_{\eta(\sigma_i)}$, then σ_i is said to be maximal.

(ii) A factorisation $\sigma_1 ... \sigma_h$ *of* σ *such that there exists an attribution function to* $\{A(p_1), \ldots, A(p_k)\}$ *is called a canonical decomposition of* σ *(such an attribution function is not necessarily unique).*
(iii) If two maximal successive sections $\sigma_1 = (s_p, \ldots, s_q)$ *and* $\sigma_2 = (s_q, \ldots, s_r)$ *of* σ *attributable respectively to* $A(p_i)$ *and to* $A(p_j)$ *are supported by the trunk* τ *of a subaraucaria of* A *with root* s_p *and if* s_u *is the node of* τ *such that* $u - p = r - q$*, then* $\sigma'_1 = (s_p, \ldots, s_u)$ *and* $\sigma'_2 = (s_u, \ldots, s_r)$ *are maximal sections of* σ *attributable respectively to* $A(p_j)$ *and to* $A(p_i)$ *which verify the equality* $\sigma_1.\sigma_2 = \sigma'_1.\sigma'_2$*.*
The replacement of $\sigma_1.\sigma_2$ *by* $\sigma'_1.\sigma'_2$ *in a canonical decomposition is called a direct pseudo-permutation.*
Each succession of direct pseudo-permutations along trunks of subaraucarias of A *is called a pseudo-permutation.*
(iv) A *is called complete for the canonical decomposition if,* $\forall h \in \{1, \ldots, k\}$*, for each one-to-one mapping* α *from* $\{1, \ldots, h\}$ *into* $\{1, \ldots, k\}$ *and each sequence* $(p'_1, \ldots, p'_h)$ *such that* $\forall i \in \{1, \ldots, h-1\}, 1 \leq p'_i \leq p_{\alpha(i)}$ *and* $p'_h = p_{\alpha(h)}$*, there exists a maximal path* σ *which admits a canonical decomposition* $\sigma_1 ... \sigma_h$ *such that* $\forall i \in \{1, \ldots, h\}, |\sigma_i| = p'_i$ *and whose attribution function* η *is such that for each* $i \in \{1, \ldots, h\}$*,* $\eta(\sigma_i) = \alpha(i)$*.*
(v) Two canonical decompositions $\sigma = \sigma_1 ... \sigma_h$ *and* $\sigma' = \sigma'_1 ... \sigma'_h$ *are called isomorphic if, for all* $i \in \{1, \ldots, h\}$*,* σ_i *and* σ'_i *have the same length and are attributable to the same elementary araucaria.*

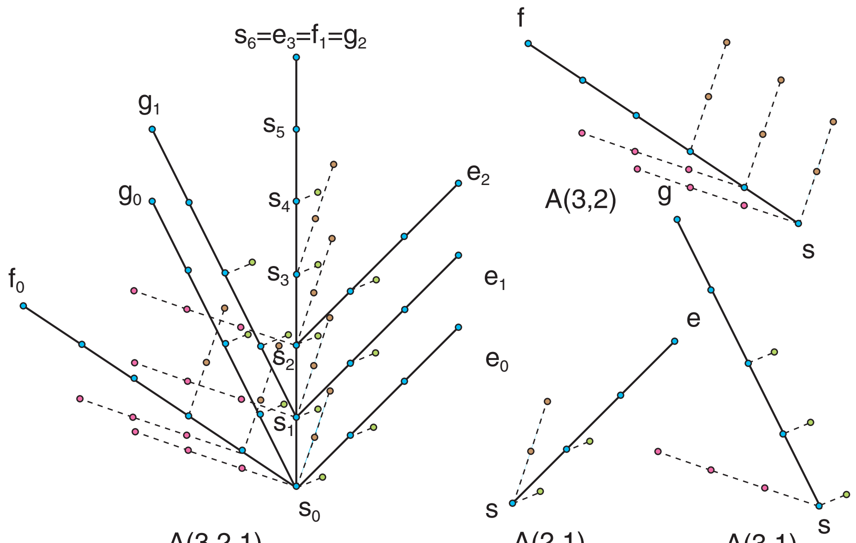


Fig. 2. The araucaria $A(3,2,1)$ has 4 subaraucarias isomorphic to $A(2,1)$, 3 subaraucarias isomorphic to $A(3,1)$ and 2 subaraucarias isomorphic to $A(3,2)$.

Theorem 1. *A directed tree A is an araucaria of type $(p_1, \dots, p_k)$ if and only if*
(i) each maximal path of A admits a canonical decomposition and this decomposition is unique up to a pseudo-permutation;
(ii) A is complete for the canonical decomposition.

Proof. (i) We follow the proof of Lemma 1 with the same notations but in addition to the induction hypotheses we assume unicity up to a pseudo-permutation and completeness. These properties are trivially verified for $k = 1$. Assume now that $k > 1$ and let $\sigma_1 = (s_0, \dots, s_q)$ be the first section of the canonical decomposition $\sigma_1 ... \sigma_h$ of a maximal path σ.
If the nodes s_{q-1} and s_{q+1} do not belong to a same trunk of a subaraucaria of A and in particular if σ_1 is not maximal, the only pseudo-permutations of the factorisation $\sigma_1 ... \sigma_h$ of σ are these of $\sigma' = \sigma_2 ... \sigma_h$ and this proves the unicity of the factorisation of σ in this case.
In the opposite case, there exists a trunk τ of a subaraucaria of A which contains the sections $\sigma_1, ..., \sigma_g$ of σ but not σ_{g+1} when $g < h$. If $g > 2$, the pseudo-permutation of $\sigma_1 . \sigma_2$ exits by Definition 2 and since, by induction, all the pairs of the set $\{\sigma_2, ..., \sigma_{g-1}\}$ if $g < h$ [resp. $\{\sigma_2, ..., \sigma_g\}$ if $g = h$] are pseudo-permutable, all the pairs of the set $\{\sigma_1, ..., \sigma_{g-1}\}$ [resp. $\{\sigma_1, ..., \sigma_g\}$] are also pseudo-permutable. This proves the unicity of the factorisation of σ up to a pseudo-permutation also in this case.
By Definition 1, $\eta(\sigma_1)$ can be any element of $\{1, \dots, k\}$ and $|\sigma_1|$ can take any value between 1 and $p_{\eta(\sigma_1)}$. By induction, it follows that A is complete for the canonical decomposition.
(ii) The converse is trivial for $k = 1$.
Assume that the converse is true for each directed tree which verifies the property for at most $k - 1$ elementary araucarias and let B be a directed tree whose maximal paths admit all a canonical decomposition relatively to the set $\{A(p_1), \dots, A(p_k)\}$ of elementary araucarias and that such a decomposition is unique up to a pseudo-permutation.
Let s_0 be the root of B. Suppose that $\sigma_1 = (s_0, \dots, s_q)$ is a section attributable to some elementary araucaria $A(p_i)$ with $i \in \{1, \dots, k\}$. For each maximal path $\sigma = (s_0, \dots, s_f)$ of B which admits a canonical decomposition whose first factor is equal to a σ_1 and is attributable to $A(p_i)$, $\sigma' = (s_q, \dots, s_f)$ admits also such a decomposition and, by the induction hypothesis, the subtree B' of B with root s_q is a directed subtree of an araucaria whose type is a subsequence of $(p_1, \dots, p_k)$ which does not contain p_i. It follows by Definition 1 that B is a directed subtree of an araucaria of type $(p_1, \dots, p_k)$.
Moreover if all directed subtrees B' of B are complete for the canonical decomposition, the same is true for B and this proves that B is an araucaria by induction on k.
Hence there exists an unique araucaria $A(p_1, \dots, p_k)$ of type $(p_1, \dots, p_k)$ up to an isomorphism. □

Definition 3. *(i) If $\sigma = (s_0, \ldots, s_f)$ is a maximal path in an araucaria A, a section $\tau = (s_i, \ldots, s_j)$ of σ is called a truncation of σ if it is supported by the trunk of A or of one of its subaraucarias and is maximal with respect to this property. A truncation τ is called terminal if its last node is a leaf of A.*
(ii) If σ is a maximal path, each factorisation $\sigma = \tau_1 ... \tau_t$ whose factors are truncations is called a decomposition into truncations.
The number of truncations is called the rank of σ.

It follows that each terminal truncation is the product of attributable maximal pseudo-permutable sections and each maximal path of an araucaria admits a decomposition into truncations and this decomposition is unique.

2.2 Shuffle Product of Elementary Araucarias

The proof of Theorem 2 below uses the last attributable section although the proof of Theorem 1 uses the first attributable section. This new characterization of the auracarias will be used in the next section.

Definition 4. *Let $A = (S, U)$ be a directed tree and $\sigma = (c_0, \ldots, c_r)$ a path of length r.*
Let, for each node s of A, $\sigma(s) = (s_0, \ldots, s_r)$ be a path isomorphic to σ such that $s_0 = s$ and $S \cap \{s_1, \ldots, s_r\} = \emptyset$ and such that, for each node t of A distinct from s, $c(s) \cap c(t) = \emptyset$.
The directed tree C obtained by connecting A with all paths $\sigma(s)$ for $s \in S$ is called the ramified directed tree of A with respect to σ and is denoted $branch(A, \sigma)$ (see Figure 1).

Lemma 2. *Let $(p_1, \ldots, p_k)$ be a sequence of strictly positive integers, i and j such that $0 < i \leq k$, $0 < j \leq k$ and $i \neq j$, A_i and A_j araucarias of types respectively $(p_{i+1}, \ldots, p_k, p_1, \ldots, p_{i-1})$ and $(p_{j+1}, \ldots, p_k, p_1, \ldots, p_{j-1})$.*
For each maximal path σ of $A'_i = branch(A_i, A(p_i))$, there exists a maximal path σ' of $A'_j = branch(A_j, A(p_j))$ such that σ and σ' admit isomorphic canonical decompositions up to a pseudo-permutation if and only if the terminal truncation τ of σ contains a maximal section which is attributable to $A(p_j)$.

Proof. The proof is omitted. □

Definition 5. *(i) Let $\sigma = (s_0, \ldots, s_h)$ and $\sigma' = (s'_0, \ldots, s'_h)$ be two paths of equal length in a graph G. Merging σ and σ' consits in merging, for all i of $\{0, \ldots, h\}$, the nodes s_i and s'_i into an unique node, and for all i in $\{0, \ldots, h-1\}$ in merging the edges (s_i, s_{i+1}) and (s'_i, s'_{i+1}) into an unique edge.*
(ii) Let k be an integer such that $1 < k$ and let $(p_1, \ldots, p_k)$ be a sequence of strictly positive integers.
$\forall i \in \{1, \ldots, k\}$, let A_i be an araucaria of type $(p_{i+1}, \ldots, p_k, p_1, \ldots, p_{i-1})$ and $A'_i = branch(A_i, A(p_i))$ and let B be the disjoint union of the directed trees $A'_1, \ldots, A'_k$.

The directed graph A obtained by merging for each couple of maximal paths (σ, σ') of B having isomorphic canonical decompositions up to a pseudo-permutation, the terminal truncations of σ and σ', is called the shuffle product of the elementary araucarias $A(p_1), \ldots, A(p_k)$.

Theorem 2. *Let k be an integer such that $k > 1$ and $(p_1, \ldots, p_k)$ any sequence of strictly positive integers.*
The shuffle product of the elementary araucarias $A(p_1), \ldots, A(p_k)$ is an araucaria of type $(p_1, \ldots, p_k)$.

Proof. (i) It is not difficult to see that the directed graph A is a directed tree.

(ii) For each $i \in \{1, \ldots, k\}$, since the araucaria A_i is of type $(p_{i+1}, \ldots, p_k, p_1, \ldots, p_{i-1})$, each maximal path σ of A_i admits a canonical decomposition into sections $\sigma_1, \ldots, \sigma_h$ attributable to two by two distinct elementary araucarias of the set $\{A(p_1),\ldots,A(p_{i-1}), A(p_{i+1}),\ldots,A(p_k)\}$ by Theorem 1 and this decomposition is unique up to a pseudo-permutation. The extremity e of σ is a leaf and, if σ_{h+1} is the path issued from e which is isomorphic to $A(p_i)$, the path $\sigma' = \sigma.\sigma_{h+1}$ in the ramified directed tree $A'_i = branch(A_i, A(p_i))$ is maximal and admits $\sigma_1 ... \sigma_h.\sigma_{h+1}$ as canonical decomposition and this decomposition is unique up to a pseudo-permutation in A'_i. By the proof of Theorem 1, A'_i is isomorphic to a directed subtree of the araucaria of type $(p_1, \ldots, p_k)$.

(iii) Since, as in the construction of A, we join together all the directed trees $A'_1,\ldots,A'_k$ and then by Lemma 2, we merge all pairs of maximal paths which admit isomorphic canonical decompositions up to a pseudo-permutation, each maximal path of A admits a canonical decomposition and this decomposition is unique up to a pseudo-permutation.
Since, for each $i \in \{1, \ldots, k\}$ the araucaria A_i is complete for the canonical decomposition, A'_i contains all maximal paths whose terminal section is attibutable to $A(p_i)$. It follows that A is complete for the canonical decomposition. By Theorem 1, A is therefore an araucaria of type $(p_1, \ldots, p_k)$. $\square$

2.3 The Size of an Araucaria

Now we introduce a family of remarkable polynomials which will be helpful for the calculation of the size of the araucarias.

Definition 6. *Let $\{X_1, \ldots, X_k\}$ be a set of k variables.*
For each $m \in \{1, \ldots, k\}$, let $\Psi_m(X_1, \ldots, X_k)$ be the polynomial sum of the products of m two by two distinct variables of the set $\{X_1, \ldots, X_k\}$ and let $\Psi_0(X_1, \ldots, X_k) = 1$.
The polynomial $\Upsilon_k(X_1, \ldots, X_k) = \sum_{m=0}^{m=k} m! * \Psi_m(X_1, \ldots, X_k)$ is called the araucaria polynomial in k variables.
The first araucaria polynomials are $\Upsilon_1(X_1) = X_1 + 1$, $\Upsilon_2(X_1, X_2) = 2X_1X_2 + X_1 + X_2 + 1$ and $\Upsilon_3(X_1, X_2, X_3) = 6X_1X_2X_3 + 2(X_1X_2 + X_2X_3 + X_3X_1) + X_1 + X_2 + X_3 + 1$.

Theorem 3. *For each cyclic permutation χ of the set $\{1, \ldots, k\}$,*

$$\sum_{i=1}^{i=k} \Upsilon_{k-1}(X_{\chi^i(1)}, \ldots, X_{\chi^i(k-1)}) * X_{\chi^i(k)} + 1 = \Upsilon_k(X_1, \ldots, X_k).$$

Proof sketch. This property results from the following equality $\sum_{i=1}^{i=k} \Psi_m(X_{\chi^i(1)}, \ldots, X_{\chi^i(k-1)}) * X_{\chi^i(k)} = (m+1) * \Psi_{m+1}(X_1, \ldots, X_k)$ ⊐

Theorem 4. *An araucaria of arity k and of type $(p_1, \ldots, p_k)$ has a size equal to $\Upsilon_k(p_1, \ldots, p_k)$ and the number of internal nodes is equal to $k! * p_1 * \ldots * p_k$.*

Proof. (i) If $k = 1$, for each strictly positive integer p_1, the araucaria $A(p_1)$ is reduced to a path of length p_1; its size is therefore $p_1 + 1 = \Upsilon_1(p_1)$.
Assume that the size of each araucaria of arity $k-1$ is given by the araucaria polynomial in $k-1$ variables and let A be an araucaria of arity k and of type $(p_1, \ldots, p_k)$. By Theorem 2, A is isomorphic to the shuffle product of the elementary araucarias $A(p_1), \ldots, A(p_k)$.
Let, for all $i \in \{1, \ldots, k\}$, A_i be an araucaria of type $(p_{i+1}, \ldots, p_k, p_1, \ldots, p_{i-1})$ and $A'_i = branch(A_i, A(p_i))$. By Definition 5, if B is the disjoint union of the directed trees $A'_1, \ldots, A'_k$, then A is obtained by merging, in B, the terminal truncations of each couple of maximal paths (σ, σ') which admit isomorphic canonical decompositions up to a pseudo-permutation.
Let, for each i of $\{1, \ldots, k\}$, V_i be the set of nodes of A'_i which do not belong to the directed subtree A_i. By Lemma 2 and the proof of Theorem 2, we can realize the merging of the terminal truncations for all couples of maximal paths (σ, σ') with respect to increasing rank.
Let σ be a maximal path of A and let τ be its terminal truncation. Since τ is a product of maximal attributable pseudo-permutable sections, there exist sections $\sigma_1, \ldots, \sigma_h$ respectively attributable to $A(p_{i_1}), \ldots, A(p_{i_h})$ where $i_1 < \ldots < i_h$. $\forall i \in \{1, \ldots, h\}$, let σ'_i be the set of nodes of σ_i distinct from the first node.
Since σ is a path of A'_i if and only if τ contains a section which is attributable to $A(p_i)$, none of the sets $\sigma \cap V_{i_1}, \ldots, \sigma \cap V_{i_h}$ is empty. These sets are not disjoint but if we replace, for each $g \in \{2, \ldots, h\}$, the part $\sigma \cap V_{i_g}$ of V_{i_g} by the set σ'_g then they become two by two disjoint.
Let $V'_1, \ldots, V'_k$ be the residual sets obtained respectively from $V_1, \ldots, V_k$ by these substitutions for all maximal path of A whose terminal truncation is not reduced to a unique attributable section. Then, for each maximal path of A, the sets $V'_1 \cap \sigma, \ldots, V'_k \cap \sigma$ which are not empty, are two by two disjoint and $(V'_1 \cap \sigma) \cup \ldots \cup (V'_k \cap \sigma)$ contains all nodes of σ distinct from the first node. It follows that the residual sets $V'_1, \ldots, V'_k$ are two by two disjoint and $V'_1 \cup \ldots \cup V'_k$ contains all nodes of A out of the root. Hence, $card(A) = 1 + \sum_{i=1}^{i=k} card(V'_i)$.
Since, for each $i \in \{1, \ldots, k\}$, each replacement of a part of V_i does not modify its cardinality, $card(V'_i) = card(V_i)$.
Moreover each node t of V_i is the extremity of an edge which belongs to a section which is attribuable to $A(p_i)$ and the set of these edges is the union of

all sections attributable to $A(p_i)$. It follows, by the induction hypothesis, that $card(V_i) = p_i * \varUpsilon_{k-1}(p_{i+1}, \ldots, p_k, p_1, \ldots, p_{i-1})$. Therefore,
$card(A) = 1 + \sum_{i=1}^{i=k} card(V_i) = 1 + \sum_{i=1}^{i=k} p_i * \varUpsilon_{k-1}(p_{i+1}, \ldots, p_k, p_1, \ldots, p_{i-1})$
and, by Theorem 3, $card(A) = \varUpsilon_k(p_1, \ldots, p_k)$.
(ii) We can prove, thanks to a double induction on k and p_k, that an araucaria of arity k and of type $(p_1, \ldots, p_k)$ has $\sum_{m=0}^{m=k-1} m! * \varPsi_m(p_1, \ldots, p_k)$ leaves. □

3 On Some Subautomatas of the Minimal Automaton of the Shuffle of a Set of Words $u_1, u_2, \ldots, u_k$

Definition 7. *(i) Let u and v be two words of the free monoid A^*.*
The language whose words are of the form $u_1v_1u_2v_2 \ldots u_mv_m$ where $u_1u_2 \ldots u_m$ is a factorisation of u, $v_1v_2 \ldots v_m$ a factorisation of v and the factors u_1 and v_m are possibly empty, is called the shuffle of the words u and v and is denoted $u \sqcup\!\sqcup v$.
(ii) If I and J are two languages of A^*, the union of the sets $u \sqcup\!\sqcup v$ for $u \in I$ and $v \in J$ is called the shuffle of the languages I and J and is denoted $I \sqcup\!\sqcup J$.
(iii) Let $u_1, \ldots, u_k$ be k words of A^*. If we assume that the shuffle K of the words $u_1, \ldots, u_{k-1}$ is defined, the language $L = K \sqcup\!\sqcup u_k$ is called the shuffle of the words $u_1, \ldots, u_k$ and is denoted

$$L = u_1 \sqcup\!\sqcup \ldots \sqcup\!\sqcup u_k.$$

Definition 8. *(i) Let $u = a_1 \ldots a_n$ be a word of length n of A^*. The partial automaton $PA(u)$ whose set of states is $\{s_0, \ldots, s_n\}$, whose transitions are (s_i, a_i, s_{i+1}) with $i \in \{0, \ldots, n-1\}$, s_0 the initial state and s_n the terminal state, is called the partial minimal automaton of u.*
(ii) Let $u_1, \ldots, u_k$ be k words of A^*, $L = u_1 \sqcup\!\sqcup \ldots \sqcup\!\sqcup u_k$, $A(L)$ the minimal automaton of L and d its initial state.
Since all words of L have the same length, $A(L)$ has an unique terminal state f and an absorbing state z ($\forall a \in A, z.a = z$).
The partial automaton $PA(L)$ obtained from $A(L)$ by deleting the absorbing state z and all transitions towards z or issued from z, is called the partial minimal automaton of L.

Definition 9. *(i) Let S_0 be the set of states of the partial minimal automaton $PA(K)$, d_0 the initial state of $PA(K)$, f_0 its unique terminal state and $u_k = a_1 \ldots a_n$.*
For each $i \in \{0, \ldots, n\}$, let $\theta_i : s \to s^{(i)}$ be an isomorphism from $PA(K)$ on an automaton $PA(K)^{(i)}$ such that the sets $S^{(i)} = \theta_i(S_0)$ are two by two disjoint.
The non-deterministic automaton $M_1(L)$ which is the disjoint union of the partial automata $PA(K)^{(0)}$, $\ldots$, $PA(K)^{(n)}$ and which admits, for each $s \in S_0$ and each $i \in \{0, \ldots, n-1\}$, $(s^{(i)}, a_{i+1}, s^{(i+1)})$ as transition, is called the shuffle product of the partial automata $PA(K)$ and $PA(u_k)$ and its denotation is

$PA(K) \sqcup\!\sqcup\, PA(u_k)$. It admits $S_1 = S_0^{(0)} \cup \ldots \cup S_0^{(n)}$ as set of states, $d_1 = d_0^{(0)}$ as initial state and $f_1 = f_0^{(n)}$ as terminal state.
Each transition of the form $(s^{(i)}, a_{i+1}, s^{(i+1)})$ is called vertical and each transition of one of the partial automata $PA(K)^{(i)}$ is called horizontal.
The non-deterministic automaton $PA(K) \sqcup\!\sqcup\, PA(u_k)$ recognizes the language L.
(ii) Let $M_2'(L)$ be the subautomaton of the automaton of parts of $PA(K) \sqcup\!\sqcup\, PA(u_k)$ generated by $d_2 = \{d_1\}$.
The partial subautomaton $M_2(L)$ of the automaton $M_2'(L)$ obtained by deleting the empty set of S_1, is called the determinization of $M_1(L) = PA(K) \sqcup\!\sqcup\, PA(u_k)$.

Definition 10. *(i) From now on we will study the following particular case: $\forall i \in \{1, \ldots, k\}$, the word u_i is of the form $u_i = b_i a^{p_i} c_i$ with strictly positive integers $p_1, \ldots, p_k$ and two by two disjoint letters of $\{b_1, \ldots, b_k\} \cup \{c_1, \ldots, c_k\}$ up to the equalities $b_1 = c_1, \ldots, b_k = c_k$.*
Such a set $(u_1, \ldots, u_k)$ is called special.
(ii) Let d be the initial state of the partial minimal automaton $PA(L)$ and let T be the set of states t such that there exist
- left factors v_1,..., v_k respectively of u_1,..., u_k such that, for all i of $\{1, ...k\}$, $1 \leq |v_i| \leq 1 + p_i$
- and a word v of $v_1 \sqcup\!\sqcup \ldots \sqcup\!\sqcup\, v_k$ such that $t = d.v$.
The partial automaton N of $PA(L)$ which admits T as set of states and the a-transitions of the form $(t, a, t.a)$ such that $t.a \in T$ as only transitions, is called the nest of a-transitions in $PA(L)$ (see Figure 3).
(iii) A state t of T is called an entry for the letter b_i if there exists a state s in $PA(L)$ and a transition (s, b_i, t) from s to t.
(iv) The definition of the nest of a-transitions in the non deterministic automata $PA(K) \sqcup\!\sqcup\, PA(u_k)$ is similar.

Lemma 3. *Let $(u_1, \ldots, u_k)$ be a special set of words and let N_0 and N be the respective nests of a-transitions of $PA(K)$ and $PA(L)$.*
If the opposite graph G_0 of the graph of the nest N_0 is a directed tree, then the opposite graph G of the graph of the nest N admits a subgraph which is isomorphic to the ramified directed tree $branch(G_0, A(p_k))$.

Proof. The proof is omitted. □

Theorem 5. *If the set $(u_1, \ldots, u_k)$ is special, the graph of the nest N of a-transitions in the automaton $PA(L)$ is the opposite of an araucaria of type $(p_1, \ldots, p_k)$*

Proof. (i) The property is trivial for $k = 1$.
Assume that, for each sequence $(p_1, \ldots, p_{k-1})$ of strictly positive integers, the graph of the nest of a-transitions in $PA(K)$ is the opposite of an araucaria of type $(p_1, \ldots, p_{k-1})$.

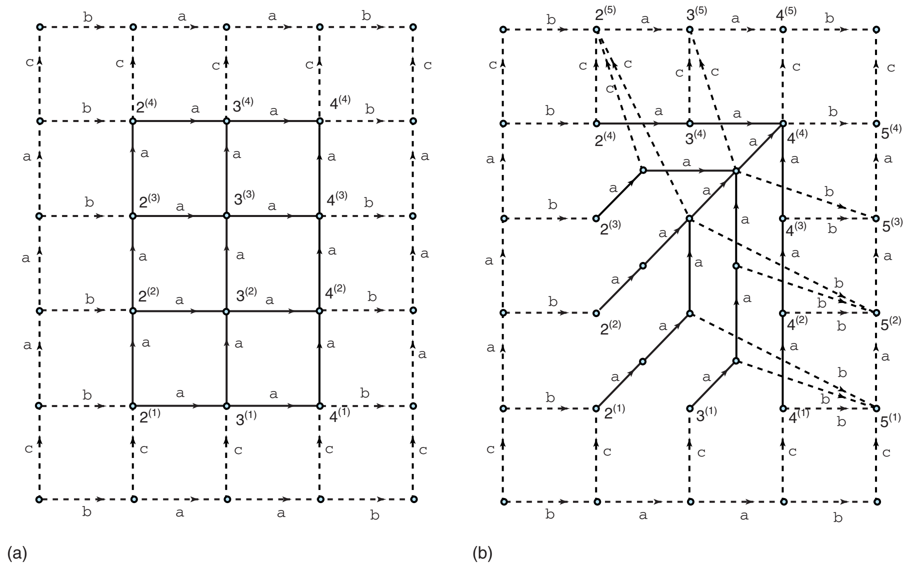


Fig. 3. Let $u_1 = baab$, N_0 the nest of $PA(u_1)$, $u_2 = caaac$ and $L = u_1 \sqcup\!\sqcup\, u_2$. (a) The nondeterministic nest $N_1 = N_0 \sqcup\!\sqcup\, PA(a^3)$ in $PA(u_1) \sqcup\!\sqcup\, PA(u_2)$; (b) The corresponding nest N in the partial minimal automaton $PA(L)$ and its subgraph which is the opposite of an araucaria of type $(2, 3)$.

By Lemma 3, the opposite graph G of the graph of the nest N contains a directed subtree A'_k which is isomorphic to the ramified directed tree $branch(A_k, A(p_k))$ where A_k is an araucaria of type $(p_1, \ldots, p_{k-1})$.
Since, for each permutation χ of $\{1, \ldots, k\}$, the languages $u_{\chi(1)} \sqcup\!\sqcup \ldots \sqcup\!\sqcup\, u_{\chi(k)}$ and $u_1 \sqcup\!\sqcup \ldots \sqcup\!\sqcup\, u_k$ are the same, the graph G contains also, for each $i \in \{1, ..., k-1\}$, a directed subtree A'_i which is isomorphic to $branch(A_i, A(p_i))$ where A_i is an araucaria of type $(p_{i+1}, \ldots, p_k, p_1, \ldots, p_{i-1})$.
Since A'_k contains all entries relative to the letter b_k, the same thing happens for the other entry letters $b_1, \ldots, b_{k-1}$. G is therefore covered by the directed trees $A'_1, \ldots, A'_k$.
(ii) For each entry e relative to a letter b_i in the nest N_0 of a-transitions in the automaton $PA(K)$, $e^{(1)}$ is simultaneously an entry for b_i and for b_k by Lemma 3. If κ is the canonical morphism from $M_2(L)$ on $PA(L)$, there exists therefore, in G, a maximal path σ from the root r of G to the node $\kappa(e^{(1)})$ and this node is common to the subtrees A'_i and A'_k. The directed trees A'_i and A'_k are therefore merged with respect to the maximal path σ. The same argument applies for each common entry of any number of letters of $\{b_1, \ldots, b_{k-1}\}$ and, permuting the words $u_1, \ldots, u_k$, to each part of $\{b_1, \ldots, b_k\}$.
First we merge the paths issued from the root which end with the unique entry common to all letters of $\{b_1, \ldots, b_k\}$, then we do the same thing for the entries common to $k-1$ letters and so forth until the common entries of two letters. The operation consists then in merging the terminal truncations.

G is therefore isomorphic to the shuffle product of the elementary araucarias $A(p_1), \ldots, A(p_k)$ and is, by Theorem 2, isomorphic to an araucaria of type $(p_1, \ldots, p_k)$. $\square$

4 Conclusion

In this paper we have investigated directed trees which appear in the construction of the minimal automaton of the shuffle of words. More results concerning this automaton have already been proved or are the object of work in progress. In particular, we hope to be able to prove that, if the set of words is special, then the size of the partial minimal automaton of the shuffle of words is maximal. The design of an optimal algorithm for the construction of this automaton is under investigation by the present authors.

References

1. Allauzen C., Calcul efficace du shuffle de k mots, Prépublication de l'Institut Gaspard Monge, 36, 2000.
2. Berstel J. and Boasson L., Shuffle factorization is unique, Prépublication de l'Institut Gaspard Monge, 16, 1999.
3. Cori R. and Perrin D., Automates et commutations partielles, *RAIRO Inf. Theor.*, **19**, 1985, 21-32.
4. Diekert V., Combinatorics of traces, *Lecture Notes in Computer Science*, **454**, Springer Verlag, 1990.
5. Eilenberg S., Automata, Languages and Machines, Academic Press, 1974.
6. Françon J., Une approche quantitative de l'exclusion mutuelle, *Theoretical Informatics and Applications*, **20**, 3 (1986) 275-289.
7. Gómez A. C. and Pin J.-E., On a conjecture of Schnoebelen, *Lecture Notes in Computer Science*, **2710** (2003), 35-54.
8. Lothaire M., Combinatorics on words, Addison-Wesley, Reading, MA, 1982.
9. Mateescu A., Shuffle of ω-words: algebraic aspects, Proceedings of STACS 98, *Lecture Notes in Computer Science*, **1373**, 150-160, Springer Verlag 1998.
10. Nivat M., Ramkumar G.D.S., Pandu Rangan C., Saoudi A., and Sundaram R., Efficient parallel shuffle recognition, *Parallel Processing Letters*, 4, 455-463, 1994.
11. Schnoebelen P., Decomposable regular languages and the shuffle operator, *EATCS Bull.*, **67**, 1999, 283-289.
12. Schott R. and Spehner J.-C., Efficient generation of commutation classes, *Journal of Computing and Information*, **2**, **1**, 1996, 1110-1132.
    Special issue: Proceedings of Eighth International Conference of Computing and Information (ICCI'96), Waterloo, Canada, June 19-22, 1996.
13. Schott R. and Spehner J.-C., Two optimal parallel algorithms on the commutation class of a word, *Theoretical Computer Science*, **324**, 107-131, 2004.
14. Spehner J.-C., Le calcul rapide des mélanges de deux mots, *Theoretical Computer Science*, 47 (1986), 181-203.
15. Zielonka W., Notes on finite asynchronous automata and trace languages, *RAIRO Inf. Theor.*, **21**, 1987, 99-135.

Author Index