How to Go Beyond Turing with P Automata: Time Travels, Regular Observer ω-Languages, and Partial Adult Halting

Rudolf Freund[1], Sergiu Ivanov[2], and Ludwig Staiger[3]

[1] Technische Universität Wien, Austria
Email: rudi@emcc.at

[2] Université Paris Est, France
Email: sergiu.ivanov@u-pec.fr

[3] Martin-Luther-Universität Halle-Wittenberg, Germany
Email: staiger@informatik.uni-halle.de

Summary. In this paper we investigate several variants of P automata having infinite runs on finite inputs. By imposing specific conditions on the infinite evolution of the systems, it is easy to find ways for going beyond Turing if we are watching the behavior of the systems on infinite runs. As specific variants we introduce a new halting variant for P automata which we call *partial adult halting* with the meaning that a specific predefined part of the P automaton does not change any more from some moment on during the infinite run. In a more general way, we can assign ω-languages as observer languages to the infinite runs of a P automaton. Specific variants of regular ω-languages then, for example, characterize the red-green P automata.

1 Introduction

Various possibilities how one can "go beyond Turing" are discussed in [11], for example, the definitions and results for red-green Turing machines can be found there. In [2] the notion of red-green automata for register machines with input strings given on an input tape (often also called *counter automata*) was introduced and the concept of *red-green P automata* for several specific models of membrane systems was explained. Via red-green counter automata, the results for acceptance and recognizability of finite strings by red-green Turing machines were carried over to red-green P automata. The basic idea of red-green automata is to distinguish between two different sets of states (red and green states) and to consider infinite runs of the automaton on finite input objects (strings, multisets); allowed to change between red and green states more than once, red-green automata can recognize more than the recursively enumerable sets (of strings, multisets), i.e., in that way we can "go beyond Turing". In the area of P systems, first attempts to do that can

be found in [4] and [18]. Computations with infinite words by P automata were investigated in [9].

In this paper, we also consider infinite runs of P automata, but in a more general way take into account the existence/non-existence of a recursive feature of the current sequence of configurations. In that way, we obtain infinite sequences over $\{0,1\}$ which we call "observer languages" where 1 indicates that the specific feature is fulfilled by the current configuration and 0 indicates that this specific feature is not fulfilled. The recognizing runs of red-green automata then correspond with ω-regular languages over $\{0,1\}$ of a specific form ending with 1^ω as observer languages. A very special observer language is $\{0,1\}^* \{1\}^\omega$ which corresponds with a very special acceptance condition for P automata which we call "partial adult halting". This special acceptance variant for P automata with infinite runs on finite multisets is motivated by an observation we make for the evolution of time lines described by P systems – at some moment, a specific part of the evolving time lines, for example, the part describing time 0, shall not change any more.

2 Definitions

We assume the reader to be familiar with the underlying notions and concepts from formal language theory, e.g., see [17], as well as from the area of P systems, e.g., see [13, 14, 15]; we also refer the reader to [25] for actual news.

2.1 Prerequisites

The set of integers is denoted by $\mathbb{Z}$, and the set of non-negative integers by $\mathbb{N}$. Given an alphabet V, a finite non-empty set of abstract symbols, the free monoid generated by V under the operation of concatenation is denoted by V^*. The elements of V^* are called strings, the empty string is denoted by λ, and $V^* \setminus \{\lambda\}$ is denoted by V^+. For an arbitrary alphabet $V = \{a_1, \ldots, a_n\}$, the number of occurrences of a symbol a_i in a string x is denoted by $|x|_{a_i}$, while the length of a string x is denoted by $|x| = \sum_{a_i \in V} |x|_{a_i}$. A (finite) multiset over a (finite) alphabet $V = \{a_1, \ldots, a_n\}$ is a mapping $f : V \to \mathbb{N}$ and can be represented by $\left\langle a_1^{f(a_1)}, \ldots, a_n^{f(a_n)} \right\rangle$ or by any string x for which $(|x|_{a_1}, \ldots, |x|_{a_n}) = (f(a_1), \ldots, f(a_n))$. The families of regular and recursively enumerable string languages are denoted by REG and RE, respectively.

2.2 Register Machines

A *register machine* is a tuple $M = (m, B, l_0, l_h, P)$, where m is the number of registers, B is a set of labels, $l_0 \in B$ is the initial label, $l_h \in B$ is the final label, and P is the set of instructions bijectively labeled by elements of B. The instructions of M can be of the following forms:

- $l_1 : (ADD(r), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$.
  Increases the value of register r by one, followed by a non-deterministic jump to instruction l_2 or l_3. This instruction is usually called *increment*.
- $l_1 : (SUB(r), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$.
  If the value of register r is zero then jump to instruction l_3; otherwise, the value of register r is decreased by one, followed by a jump to instruction l_2. The two cases of this instruction are usually called *zero-test* and *decrement*, respectively.
- $l_h : HALT$. Stops the execution of the register machine.

A *configuration* of a register machine is described by the contents (i.e., by the number stored in the register) of each register and by the current label, which indicates the next instruction to be executed. Computations start by executing the instruction l_0 of P, and terminate with reaching the HALT-instruction l_h.

In order to deal with strings, this basic model of register machines can be extended by instructions for reading from an input tape and writing to an output tape containing strings over an input alphabet T_{in} and an output alphabet T_{out}, respectively:

- $l_1 : (read(a), l_2)$, with $l_1 \in B \setminus \{l_h\}$, $l_2 \in B$, $a \in T_{in}$.
  Reads the symbol a from the input tape and jumps to instruction l_2.
- $l_1 : (write(a), l_2)$, with $l_1 \in B \setminus \{l_h\}$, $l_2 \in B$, $a \in T_{out}$.
  Writes the symbol a on the output tape and jumps to instruction l_2.

Such a register machine working on strings often is also called a *counter automaton*, and we write $M = (m, B, l_0, l_h, P, T_{in}, T_{out})$. If no output is written, we omit T_{out}.

As is well known (e.g., see [12]), for any recursively enumerable set of natural numbers there exists a register machine with (at most) three registers accepting the numbers in this set. Counter automata, i.e., register machines with an input tape, with two registers can simulate the computations of Turing machines and thus characterize RE. All these results are obtained with deterministic register machines, where the ADD-instructions are of the form $l_1 : (ADD(r), l_2)$, with $l_1 \in B \setminus \{l_h\}$, $l_2 \in B$, $1 \leq j \leq m$.

2.3 The Arithmetical Hierarchy

The Arithmetical Hierarchy (e.g., see [3]) is usually developed with the universal ($\forall$) and existential ($\exists$) quantifiers restricted to the integers. Levels in the Arithmetical Hierarchy are labeled as Σ_n if they can be defined by expressions beginning with a sequence of n alternating quantifiers starting with $\exists$; levels are labeled as Π_n if they can be defined by such expressions of n alternating quantifiers that start with $\forall$. Σ_0 and Π_0 are defined as having no quantifiers and are equivalent. Σ_1 and Π_1 only have the single quantifier $\exists$ and $\forall$, respectively. We only need to consider alternating pairs of the quantifiers $\forall$ and $\exists$ because two quantifiers of the same type occurring together are equivalent to a single quantifier.

3 Time Travel P Systems

In the most general case, we can think of P systems as devices manipulating multisets in a hierarchical membrane structure. The membranes can have labels and polarizations both eventually changing with the application of rules. Membranes may be divided, generated or deleted. Together with the division or the generation of a new membrane the whole contents of another membrane may be copied. For a general framework of P systems we refer to [7].

Usually, configurations in P systems (and other systems like Turing machines) evolve step by step through time, see Figure 1.

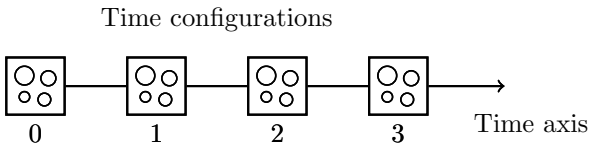


Fig. 1. Standard time line evolution.

Without time travel option, we need only consider the evolution of the system on one time axis from time n to time $n+1$. The situation becomes more difficult if we follow the idea of parallel worlds (*time axes*), which means that we have another time dimension, described by the vertical evolution in Figure 2, i.e., the time configurations at time n may be altered depending on future evolutions.

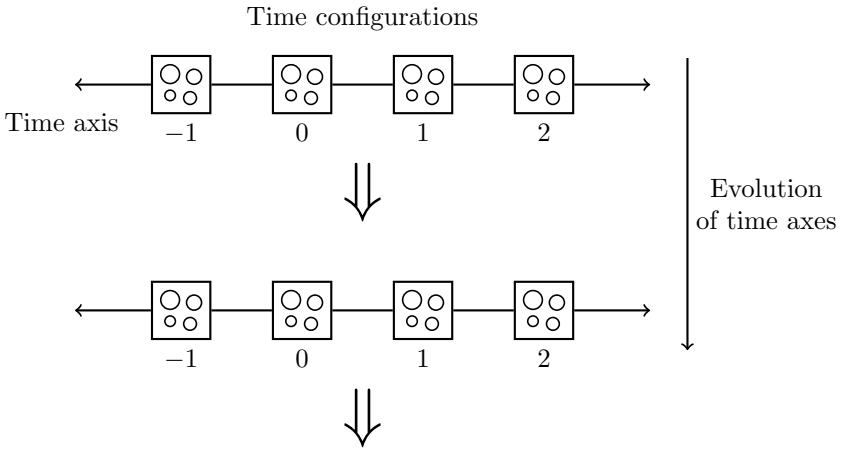


Fig. 2. Time lines evolution.

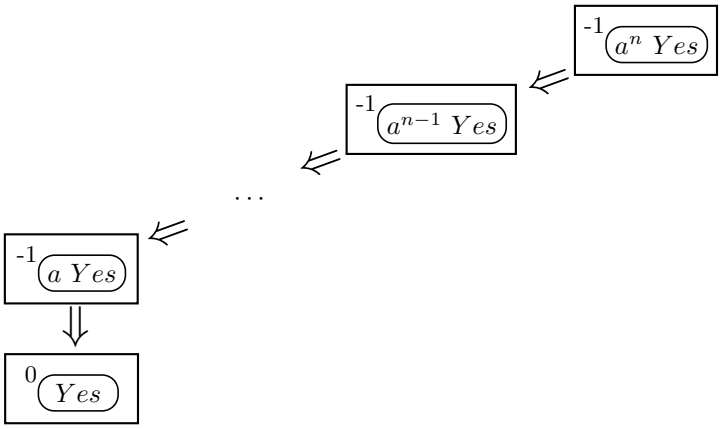


Fig. 3. Sending back an answer from time n to time 0.

For example, we can consider membrane systems with polarizations assigned to the membranes. The usual polarization of the whole time configuration in the normal case is $+1$, indicating that the evolution of the membrane(s) goes from time configuration n to time configuration $n+1$. Now assume we allow polarization -1 indicating that the corresponding membrane evolves from time configuration n to time configuration $n-1$. Having kept trace of the number of computation steps, e.g., by the multiplicity of a specific object a, we are able to send back information – like the answer *yes* to a question we have posed at time 0 which then is sent back to time configuration 0, i.e., to the time we have posed the question. In that way, on a specific time line we can have answers to questions in zero time, see Figure 3.

During its travel through the time back, the time capsule with polarization -1 can be assumed not to be affected by the other membranes in the intermediate time configurations. Obviously, this restriction can be alleviated for even more complex systems.

Putting a new skin membrane around all the current time configurations of one time axis, we again obtain a conventional evolution model, yet now with a vertical time evolution as depicted in Figure 4. The only assumption we have to do for making this variant possible is that at the beginning only a finite number of time configurations exists (in fact, we usually will start with the time configuration at time 0).

3.1 Partial Adult Halting

Going back to the time travel model of Figure 2 the question that arises is what kind of results we may obtain and how. For example, given a specific input in time configuration 0, we may request that from some moment on this time configuration becomes stable, i.e., it is not changed any more (by time capsules arriving there).

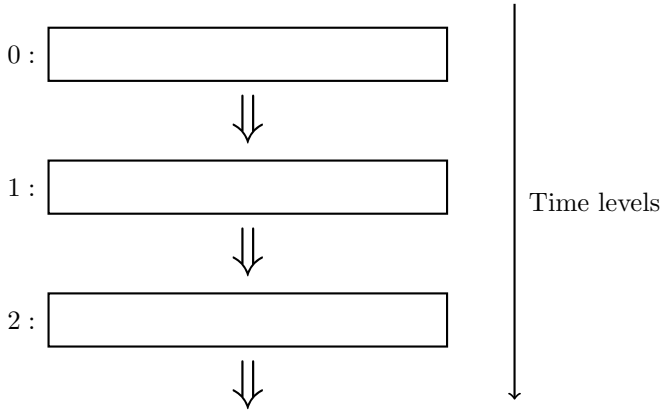


Fig. 4. Conventional Evolution Model.

So the specific feature an external observer would see is that the time configuration at time 0 is not changing any more starting from some specific time line at level tl_0 on, i.e., for all time levels $tl \geq tl_0$ the *time configuration at time* 0 *stays stable*.

With respect to the situation described in Figure 4 this means that one specific part (one membrane and all its contents) does not change any more.

In that way we obtain a new variant of a halting condition in P systems which we call *partial adult halting*:

adult halting:

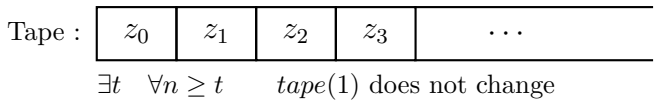
means that the configuration does not change any more

partial:

we only look at some part of the configuration

3.2 Partial Adult Halting for Turing Machines

The idea of partial adult halting can also be applied to Turing machines:



On tape cell 1 we want to obtain an "answer" whether the given input word is accepted – 1 – or not – 0. We first put 0 there, and if the computation ends saying "accept" we go back to tape cell 1 and write 1 there. Hence, with looking to infinity in that way we obtain a "decider" for recursively enumerable languages.

Generation of Complements of Recursively Enumerable Languages

Another example based on a similar idea as described above shows how to generate the complement of an arbitrary recursively enumerable language L.

In this case, we use the model of a generating Turing machine with output tape, and a string is said to be generated by the Turing machine M if from some moment of the computation the output tape is not changed anymore.

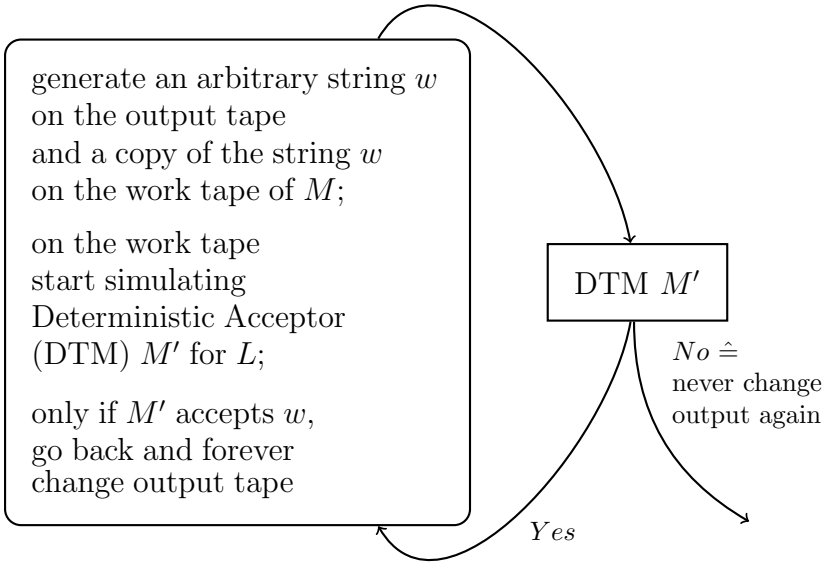


Fig. 5. Generation of the complements of a recursively enumerable language L.

4 Variants of P Automata

In this section, we shortly describe some variants of P automata.

4.1 The Basic Model of P Automata with Antiport Rules

The basic model of P automata as introduced in [6] and in a similar way in [8] is based on antiport rules, i.e., on rules of the form u/v, which means that the multiset u goes out through the membrane and v comes in instead.

A *P automaton (with antiport rules)* is a construct

$$\Pi = (O, T, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m) \text{ where}$$

- O is the alphabet of *objects*;
- $T \subset O$ is the alphabet of *terminal objects*;
- μ is the hierarchical membrane structure, with the membranes uniquely labeled by the numbers from 1 to m;
- $w_i \in (O \setminus T)^*$, $1 \leq i \leq m$, is the *initial multiset* in membrane i;
- R_i, $1 \leq i \leq m$, is a finite set of *antiport rules* assigned to membrane i.

Given a multiset of terminal symbols in the skin membrane 1, it is usually accepted by Π via a halting computation.

Now consider the situation of partial adult halting for a P automaton

$$\Pi = (O, T, [_1[_2\ ]_2]_1, q_0, n, R_1, R_2)$$

which – with the input multiset in addition given in the skin membrane – simulates, in a deterministic way, a register machine defining a recursively enumerable set L of multisets (see [12]), by the rules in R_1. If the computation stops in the final state q_h, i.e., the multiset is accepted, we add the rules q_h/y and n/n in R_1. R_2 only contains the rule n/y. In case the multiset is accepted, n in the second membrane is replaced by y, while the rule n/n in R_1 guarantees an infinite computation. In case the input multiset is not accepted, the register machine already guarantees an infinite computation by the simulating P automaton, too. Hence, as in the case of the Turing machine with partial adult halting we get a "decider" for L, with the result from some moment on to be found in membrane 2.

4.2 P Automata with Anti-Matter

In *P automata with anti-matter*, for each object a we may have its anti-matter object a^-. If an object a meets its anti-matter object a^-, then these two objects annihilate each other, which corresponds to the application of the cooperative erasing rule $aa^- \to \lambda$. In the following, we shall only consider the variant where these annihilation rules have weak priority over all other rules, which allows for a deterministic simulation of deterministic register machines, see [1].

A *P automaton with anti-matter* is a construct

$$\Pi = (O, T, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m) \text{ where}$$

- O is the alphabet of *objects*;

- $T \subset O$ is the alphabet of *terminal objects*;
- μ is the hierarchical membrane structure, with the membranes uniquely labeled by the numbers from 1 to m;
- $w_i \in (O \setminus T)^*$, $1 \leq i \leq m$, is the *initial multiset* in membrane i;
- R_i, $1 \leq i \leq m$, is a finite set of

  non-cooperative rules: are rules of the form $u \to v$ where $u \in O$ and $v \in (O \times \{here, in, out\})^*$;

  matter/anti-matter annihilation rules: are cooperative rules of the form $aa^- \to \lambda$, i.e., the matter object a and its anti-matter object a^- annihilate each other, and these annihilation rules have weak priority over all other rules.

With the target indications $\{here, in, out\}$ we can leave an object in the current membrane ($here$), whereas with $\{in\}$ we send it into an inner membrane and with $\{out\}$ we send it into the surrounding membrane region.

In a similar way as in the preceding subsection we may consider the situation of partial adult halting for a P automaton

$$\Pi = (O, T, [_1[_2\ ]_2]_1, q_0, n, R_1, R_2)$$

where following the proof from [1] the register machine actions are simulated in the skin membrane; if the input multiset is accepted, by using the rules $q_h \to (f, here)(n^-, in), f \to f$, we obtain an infinite computation with the contents of membrane 2 being empty indicating the acceptance, as by the annihilation rule $nn^- \to \lambda$ the original object n is annihilated.

5 Red-Green Automata

In general, a red-green automaton M is an automaton whose set of internal states Q is partitioned into two subsets, Q_{red} and Q_{green}, and M operates without halting. Q_{red} is called the set of "red states", Q_{green} the set of "green states". Moreover, we shall assume M to be deterministic, i.e., for each configuration there exists exactly one transition to the next one.

5.1 Red-Green Turing Machines

Red-green Turing machines, see [11], can be seen as a type of ω-Turing machines on finite inputs with a recognition criterion based on some property of the set(s) of states visited (in)finitely often, in the tradition of ω-automata (see [9]), i.e., we call an infinite run of the Turing machine M on input w *recognizing* if and only if

- no red state is visited infinitely often and
- some green states (one or more) are visited infinitely often.

A set of strings $L \subset \Sigma^*$ is said to be *accepted* by M if and only if the following two conditions are satisfied:

(a) $L = \{w \mid w \text{ is recognized by } M\}$.
(b) For every string $w \notin L$, the computation of M on input w eventually stabilizes in red; in this case w is said to be *rejected*.

The phrase "mind change" is used in the sense of changing the color, i.e., changing from red to green or vice versa.

The following results were established in [11]:

Theorem 1. *A set of strings L is recognized by a red-green Turing machine with one mind change if and only if $L \in \Sigma_1$, i.e., if L is recursively enumerable.*

Theorem 2. *(Computational power of red-green Turing machines)*

(a) Red-green Turing machines recognize exactly the Σ_2-sets of the Arithmetical Hierarchy.
(b) Red-green Turing machines accept exactly those sets which simultaneously are Σ_2- and Π_2-sets of the Arithmetical Hierarchy.

5.2 Red–Green Register Machines

In [2], similar results as for red-green Turing machines were shown for red-green counter automata and register machines, respectively.

As it is well-known folklore, e.g., see [12], the computations of a Turing machine can be simulated by a counter automaton with (only two) counters; in this paper, we will rather speak of a register machine with (two) registers and with string input. As for red-green Turing machines, we can also color the "states", i.e., the labels, of a register machine $M = (m, B, l_0, l_h, P, T_{in})$ by the two colors red and green, i.e., partition its set of labels B into two disjoint sets B_{red} (red "states") and B_{green} (green "states"), and we then write $RM = (m, B, B_{red}, B_{green}, l_0, P, T_{in})$, as we can omit the halting label l_h.

The following two lemmas were proved in [2]; the step from red-green Turing machines to red-green register machines is important for the succeeding sections, as usually register machines are simulated when proving a model of P systems to be computationally complete. Therefore, in the following we always have in mind this specific relation between red-green Turing machines and red-green register machines when investigating the infinite behavior of specific models of P automata, as we will only have to argue how red-green register machines can be simulated.

Lemma 1. *The computations of a red-green Turing machine TM can be simulated by a red-green register machine RM with two registers and with string input in such a way that during the simulation of a transition of TM leading from a state p with color c to a state p' with color c' the simulating register machine uses instructions with labels ("states") of color c and only in the last step of the simulation changes to a label ("state") of color c'.*

Lemma 2. *The computations of a red-green register machine RM with an arbitrary number of registers and with string input can be simulated by a red-green Turing machine TM in such a way that during the simulation of a computation step of RM leading from an instruction with label ("state") p with color c to an instruction with label ("state") p' with color c' the simulating Turing machine stays in states of color c and only in the last step of the simulation changes to a state of color c'.*

As an immediate consequence, the preceding two lemmas yield the characterization of Σ_2 and Π_2 by red-green register machines as Theorem 2 does for red-green Turing machines, see [2]:

Theorem 3. *(Computational power of red-green register machines)*

(i) A set of strings L is recognized by a red-green register machine with one mind change if and only if $L \in \Sigma_1$, i.e., if L is recursively enumerable.
(ii) Red-green register machines recognize exactly the Σ_2-sets of the Arithmetical Hierarchy.
(iii) Red-green register machines accept exactly those sets which simultaneously are Σ_2- and Π_2-sets of the Arithmetical Hierarchy.

5.3 Red-Green P Automata

As it was shown in [2], P automata with antiport rules and with anti-matter can simulate the infinite computations of any red-green register machine, even with a clearly specified finite set of "states" having the same color as the corresponding labels ("states") of the instructions of the red-green register machine.

Hence, as a consequence, similar results as for red-green Turing machines also hold for red-green P automata with antiport rules and with anti-matter. From the results shown in [2] we therefore infer:

Theorem 4. *(Computational power of red-green P automata)*

(i) A set of multisets L is recognized by a red-green P automaton (with antiport rules, with anti-matter) with one mind change if and only if L is recursively enumerable.
(ii) Red-green P automata (with antiport rules, with anti-matter) recognize exactly the Σ_2-sets.
(iii) Red-green P automata (with antiport rules, with anti-matter) accept exactly those sets which simultaneously are Σ_2- and Π_2-sets of the Arithmetical Hierarchy.

6 Observer Languages

An observer language for infinite computations is an ω-language over $\{0,1\}$ where 1 indicates that a specific feature of the current configuration in the infinite computation sequence is fulfilled and 0 indicates that this specific feature of the current configuration is not fulfilled.

6.1 Expressing Partial Adult Halting as Observer Language

If we define the specific feature to be that no rule is applicable in the specified "observed" membrane, then acceptance by partial adult halting can be described by the (regular) ω-language $\{0,1\}^*\{1\}^\omega$.

6.2 Expressing Recognition by Red-Green P Automata Using Observer Languages

As observer languages for infinite computations in red-green P automata we again use ω-languages over $\{0,1\}$ where now 1 indicates that we will have to apply a green multiset of rules to the current configuration in the infinite computation sequence and 0 indicates that we will have to apply a red multiset of rules to the current configuration.

So for recognizing a language from RE we use the the ω-language $\{0\}^+\{1\}^\omega$, for a language from *co-RE* we use the the ω-language $\{0\}\{1\}^\omega$.

The corresponding regular ω-languages for the recognition by red-green automata (Turing machines, P automata) with multiple mind-changes are described as follows:

exactly $2k+1$ mind-changes, $k \geq 0$: $\{0\}^+\big(\{1\}^+\{0\}^+\big)^k\{1\}^\omega$

at most $2k+1$ mind-changes, $k \geq 0$: $\bigcup_{i=0}^{k}\{0\}^+\big(\{1\}^+\{0\}^+\big)^i\{1\}^\omega$

The upper bound for languages recognized by red-green P automata (with antiport rules, with anti-matter) with k mind-changes for some $k \geq 0$ is Σ_2, see [2].

These results will be refined in the next section.

7 Recognition Using Regular Observer Languages

In this section we investigate which languages are recognized by red-green P automata using observer languages defined by finite automata. This class of ω-languages defined by finite automata is well-understood and has widely been investigated (see [16, 21, 23, 24]). We follow the line of [20] where for Turing machines infinite computations accepting finite words were investigated in detail (see also [5]). In this paper a word w was accepted by a Turing machine when the sequence

$(s_i)_{i\in\mathbb{N}}$ of states the machine runs through during its accepting process fulfills certain simple conditions known from the acceptance of ω-languages. This can be seen as w to be accepted if the observed state sequence $(s_i)_{i\in\mathbb{N}}$ belongs to a certain (regular) observer language. We have to point out that usually the notion *acceptance* is used here instead of the notion *recognition* as used by van Leeuwen and Wiedermann for the red-green Turing machines.

7.1 Observer Languages of the form $W \cdot \{1\}^\omega$ with $W \in REG$

The observer languages in Section 6 all were of the form $W\cdot\{1\}^\omega$ where $W \subseteq \{0,1\}^*$ is a regular language. In this section we investigate which languages can be accepted by red-green P automata using observer languages of this form. Here we follow the line of the papers [20] and [11] where the influence of regular observer languages on the acceptance and recognition, respectively, behavior of Turing machines was investigated.

To this end we use the following theorem which follows from a general classification of regular ω-languages (see [19, 22] and also the survey [21]).

Theorem 5. *If $F \subseteq \{0,1\}^\omega$ is a regular ω-language, then*

1. *F is in the Boolean closure of Σ_2, and*
2. *if $F \in \Sigma_2 \cap \Pi_2$, then F is in the Boolean closure of Σ_1.*

Since every regular $F \subseteq \{0,1\}^*\cdot\{1\}^\omega$ as a countable set is in Σ_2, we immediately obtain the following.

Corollary 1. *If $W \subseteq \{0,1\}^*$ is a regular language then $W \cdot \{1\}^\omega$ satisfies one of the following conditions:*

1. *$W \cdot \{1\}^\omega \in \Sigma_2 \setminus \Pi_2$, or*
2. *$W \cdot \{1\}^\omega$ is a Boolean combination of ω-languages in Σ_1.*

Remark 1. In the second case we can obtain an even sharper result:

$$W \cdot \{1\}^\omega = \bigcup\nolimits_{i=0}^{k} (W_i \cdot \{0,1\}^\omega \setminus V_i \cdot \{0,1\}^\omega)$$

for suitable $k \in \mathbb{N}$ and *regular* languages $W_i, V_i \subseteq \{0,1\}^*$, $0 \le i \le k$. In particular, this is true for the ω-languages representing a bounded number of mind-changes from Subsection 6.2:

$$\bigcup\nolimits_{i=0}^{k} \{0\}^+ \big(\{1\}^+\{0\}^+\big)^i \{1\}^\omega =$$

$$\bigcup\nolimits_{i=0}^{k} \Big(\{0\}^+\big(\{1\}^+\{0\}^+\big)^i\{1\} \cdot \{0,1\}^\omega \setminus \{0\}^+\big(\{1\}^+\{0\}^+\big)^i\{1\}^+\{0\} \cdot \{0,1\}^\omega\Big)$$

From Corollary 1 we immediately infer:

Theorem 6. *Let L be recognized by a red-green P automaton (with antiport rules, with anti-matter) using an observer language $W \cdot \{1\}^\omega$ where $W \subseteq \{0,1\}^*$ is regular.*

1. *Then $L \in \Sigma_2$.*
2. *If $W \cdot \{1\}^\omega = \bigcup_{i=0}^{k}(F_i \setminus E_i)$ is a Boolean combination of ω-languages $F_i, E_i \in \Sigma_1$, $0 \leq i \leq k$, then $L = \bigcup_{i=0}^{k}(K_i \setminus L_i)$ where $K_i, L_i \in RE$, $0 \leq i \leq k$.*

The converse of Theorem 6 is also true. In particular, it shows that we can restrict ourselves to the observer languages of Subsections 6.1 and 6.2.

Theorem 7. *Let $L \in \Sigma_2$.*

1. *Then L is recognized by a red-green P automaton Π using the observer language $\{0,1\}^* \cdot \{1\}^\omega$, i.e., L is accepted by Π by partial adult halting.*
2. *Let $L = \bigcup_{i=0}^{k}(K_i \setminus L_i)$ where $K_i, L_i \in RE$, $0 \leq i \leq k$. Then there exists a red-green P automaton which recognizes L using an observer language with a bounded number of mind-changes.*

7.2 Regular Observer Languages

Admitting all regular ω-languages as observer languages extends the range of recognizable languages. In view of Theorem 5 we obtain a result extending what was shown in Theorem 6.

Theorem 8. *Let L be recognized by a red-green P automaton using an observer language $F \subseteq \{0,1\}^\omega$. Then*

1. *if F is a Boolean combination of ω-languages $F_i, E_i \in \Sigma_2$, $0 \leq i \leq k$, then $L = \bigcup_{i=0}^{k}(K_i \setminus L_i)$ where $K_i, L_i \in \Sigma_2$, $0 \leq i \leq k$,*
2. *if $F \in \Sigma_2$, then $L \in \Sigma_2$,*
3. *if $F \in \Pi_2$, then $L \in \Pi_2$, and*
4. *if F is regular and $F \in \Sigma_2 \cap \Pi_2$, then $L = \bigcup_{i=0}^{k}(K_i \setminus L_i)$ where $K_i, L_i \in RE$, $0 \leq i \leq k$.*

The converse of Theorem 8 is also true:

Theorem 9. *Let L be a Boolean combination of languages in Σ_2. Then L is recognized by a red-green P automaton using a regular observer language $F \subseteq \{0,1\}^\omega$.*

8 Conclusion

In this paper we have investigated the computational power of P automata working with infinite runs on finite input multisets. With regular observer languages $W \cdot \{1\}^\omega$, $W \in REG$, we obtain the Σ_2-sets, the same as with red-green P automata. Moreover, the Σ_2-sets are already obtained by the special observer language $\{0,1\}^* \cdot \{1\}^\omega$, which corresponds to the special acceptance condition of *partial adult halting*.

References

1. A. Alhazov, B. Aman, R. Freund: P Systems with Anti-Matter. In: [10], 66–85.
2. B. Aman, E. Csuhaj-Varjú, R. Freund: Red-Green P Automata. In: [10], 139–157.
3. P. Budnik: *What Is and What Will Be.* Mountain Math Software, 2006.
4. C.S. Calude, Gh. Păun: Bio-steps Beyond Turing. *Biosystems* **77** (2004), 175–194.
5. C.S. Calude, L. Staiger: A note on accelerated Turing machines. *Math. Structures Comput. Sci.* **20** (6) (2010), 1011–1017.
6. E. Csuhaj-Varjú, Gy. Vaszil: P Automata or Purely Communicating Accepting P Systems. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): *Membrane Computing, International Workshop, WMC-CdeA 2002, Curtea de Argeş, Romania, August 19–23, 2002, Revised Papers.* Lecture Notes in Computer Science **2597**, Springer, 2003, 219–233.
7. R. Freund, I. Pérez-Hurtado, A. Riscos-Núñez , S. Verlan: A formalization of membrane systems with dynamically evolving structures. *International Journal of Computer Mathematics* **90** (4) (2013), 801-815.
8. R. Freund, M. Oswald: A Short Note on Analysing P Systems. *Bulletin of the EATCS* **78**, 2002, 231–236.
9. R. Freund, M. Oswald, L. Staiger: ω-P Automata with Communication Rules. *Workshop on Membrane Computing, 2003*, Lecture Notes in Computer Science **2933**, Springer, 2004, 203–217.
10. M. Gheorghe, G. Rozenberg, A. Salomaa, P. Sosík, C. Zandron: 15th International Conference, CMC 2014, Prague, Czech Republic, August 20-22, 2014, Revised Selected Papers. Lecture Notes in Computer Science **8961**, Springer, 2014.
11. J. van Leeuwen, J. Wiedermann: Computation as an Unbounded Process. *Theoretical Computer Science* **429** (2012), 202–212.
12. M. L. Minsky: *Computation: Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1967.
13. Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences* **61** (1) (2000), 108–143 (and Turku Center for Computer Science-TUCS Report 208, November 1998, `www.tucs.fi`).
14. Gh. Păun: *Membrane Computing. An Introduction.* Springer, 2002.
15. Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.
16. D. Perrin and J.-É. Pin. *Infinite Words*, vol. 141 of *Pure and Applied Mathematics.* Elsevier, Amsterdam, 2004.
17. G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages,* 3 volumes. Springer, 1997.

18. P. Sosík, O. Valík: On Evolutionary Lineages of Membrane Systems. In: R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): *6th International Workshop, WMC 2005, Vienna, Austria, July 18-21, 2005, Revised Selected and Invited Papers*. Lecture Notes in Computer Science **3850**, Springer, 2006, 67–78.
19. L. Staiger: Finite-state ω-languages. *J. Comput. System Sci.* **27** (3) (1983), 434–448.
20. L. Staiger: ω-computations on Turing machines and the accepted languages. In: L. Lovász, E. Szemerédi (Eds.): *Theory of Algorithms*, Coll. Math. Soc. Janos Bolyai No.44, North Holland, Amsterdam, 1986, 393–403.
21. L. Staiger: ω-languages. In: [17], vol. 3, 339–387.
22. L. Staiger, K. Wagner: Automatentheoretische und automatenfreie Charakterisierungen topologischer Klassen regulärer Folgenmengen. *Elektron. Informationsverarb. Kybernetik* **10** (7) (1974), 379–392.
23. W. Thomas: Automata on infinite objects. In: J. van Leeuwen (Ed.): *Handbook of Theoretical Computer Science*, vol. B, pages 133–192. North Holland, Amsterdam, 1990.
24. K. Wagner: On ω-regular sets. *Inform. and Control*, 43 (2) (1979), 123–177.
25. The P Systems Website: `http://ppage.psystems.eu`.