Computationally Universal P Systems without Priorities: Two Catalysts are Sufficient

Rudolf FREUND
Department of Computer Science, Technische Universität Wien
Favoritenstr. 9, A-1040 Wien, Austria
email: rudi@emcc.at

Lila KARI
The University of Western Ontario, London, ON, Canada, N6A 5B7
email: kari@csd.uwo.ca

Marion OSWALD
Department of Computer Science, Technische Universität Wien
Favoritenstr. 9, A-1040 Wien, Austria
email: marion@emcc.at

Petr SOSÍK
Institute of Computer Science, Silesian University, Opava, Czech Republic
and Department of Computer Science
The University of Western Ontario, London, ON, Canada, N6A 5B7
email: sosik@csd.uwo.ca

Abstract. The original model of P systems with symbol objects introduced by Gheorghe Păun was shown to be computationally universal in [12], provided that catalysts and priorities of rules are used. By reduction via register machines in [18] it was shown that the priorities may be omitted from the model without loss of computational power. In [6] several variants of P systems with catalysts (but without priorities) and the number of catalysts needed for these specific variants to be computationally universal were investigated and it was shown that for the classical model of P systems with the minimal number of two membranes the number of catalysts can be reduced from six (as proved in [18]) to five; using the idea of final states (as introduced in [2] for P automata) the number of catalysts could even be reduced to four. In this paper we are able to reduce the number of catalysts again: Two catalysts are already sufficient. For P systems with external output or extended P systems we even need only one membrane and two catalysts. For the (purely) catalytic systems considered by Ibarra only three (instead of seven, see [9]) catalysts are already enough. Moreover, we show that two catalysts are enough even with only one membrane for P systems generating or accepting string languages, whereas three catalysts are sufficient again for the purely catalytic variants with only one membrane, too.

1 Introduction

In the original paper introducing membrane systems (P systems) in [12] as a symbol manipulating model catalysts as well as priority relations on the rules were used to prove them to be computationally universal; in [18] it was shown that a priority relation on the rules is not necessary to obtain this universality result. In [6] the number of catalysts was reduced by one for the variants of P systems with two membranes considered there; moreover, the number of catalysts could even be reduced by one more when considering computations reaching some finitely specified final states as in the model of P automata introduced in [2] instead of halting computations. We now will show that even two catalysts are already sufficient for all these variants.

If instead of putting the result of a computation into an ouput membrane we send the result of a computation out into the environment, we obtain P systems with external output for which two catalysts in only one membrane are already enough. In extended P systems we specify a terminal alphabet and only consider the terminal symbols contained in the skin membrane at the end of a successful computation (in effect this means that we ignore the catalysts, which of course can never be eliminated); again the skin membrane and two catalysts are sufficient.

In [9] Ibarra introduced (purely) catalytic P systems there claiming that seven catalysts are sufficient if we only allow rules with catalysts; here we show that even three catalysts are already enough. In addition, in this paper we also consider P systems with external output (see [15]) as string generating devices as well as P systems accepting string languages (P automata first were introduced in [2] as string accepting devices); we prove that every recursively enumerable string language can be generated/accepted by such P systems (computed by halting or by final state) with two catalysts in only one membrane; moreover, we also introduce the purely catalytic variants of these P systems generating/accepting string languages and we prove that we only need three catalysts in one membrane.

In the following section, after some prerequisites from formal language theory, we give a precise definition of the model of register machines used in the subsequent proofs; moreover, we describe a special variant of counter automata that we use for proving our results about P systems generating/accepting string languages. Then we define the specific variants of P systems considered in this paper. In the further parts of the paper we show how we can reduce the number of catalysts in P systems with specific stopping conditions by using new proof techniques for simulating register machines: We first prove our main results showing that every partial recursive function $f : \mathbf{N}_0^\alpha \to \mathbf{N}_0^\beta$ for vectors of non-negative integers can be computed by a (halting) P system or by a P system with final states with two membranes and (at most) $\alpha + 2$ catalysts; from these more general results we then derive the universality results for P systems generating and accepting sets of (vectors of) non-negative integers. Using well-known results for two-counter automata we prove our universality results for P systems generating/accepting string languages. A short summary finally concludes the paper.

2 Definitions

For well-known notions and basic results from the theory of formal languages, the reader is referred to [3] and [17]. We only give some basic notations first. For an alphabet V, by V^* we denote the free monoid generated by V under the operation of concatenation; the *empty string* is denoted by λ, and $V^* \setminus \{\lambda\}$ is denoted by V^+. Any subset of V^* (V^+) is called a (λ-*free) language*. Two languages $L, L' \subseteq V^*$ are considered to be equal if $L \setminus \{\lambda\} = L' \setminus \{\lambda\}$. Moreover, by $\mathbf{N}_0$ we denote the set of non-negative integers and by $N_0^{\beta} RE$ we denote the family of recursively enumerable sets of β-vectors $(y_1, ..., y_\beta)$ of non-negative integers. Two sets of β-vectors are considered to be equal if they only differ at most by the zero-vector $(0, ..., 0)$.

Let $m \geq 2$ and let k, l be two positive integers not greater than m; then we define:

$$l \ominus_m k := \begin{cases} l - k & \text{for } l > k \\ l - k + m & \text{for } l \leq k \end{cases}$$

2.1 Register Machines and Counter Automata

In this subsection we briefly recall the concept of Minsky's register machine (e.g., see [11]). Such an abstract machine uses a finite number of registers for storing arbitrarily large non-negative integers and runs a program consisting of numbered instructions of various simple types. Several variants of the machine with different number of registers and different instruction sets were shown to be computationally universal (e.g., see [11] for some original definitions and proofs as well as [5], [7] and [8] for the definitions and results we use in this paper).

An *n-register machine* is a construct

$$M = (n, P, i, h)$$

where

- n is the number of registers,
- P is a set of labelled instructions of the form $j : (op(r), k, l)$, where $op(r)$ is an operation on register r of M, j, k, l are labels from the set $Lab(M)$ (which numbers the instructions of the program of M represented by P),
- i is the initial label, and
- h is the final label.

The machine is capable of the following instructions:

$(A(r), k, l)$ Add one to the contents of register r and proceed to instruction k or to instruction l; in the deterministic variants usually considered in the literature we demand $k = l$.

$(S(r), k, l)$: If register r is not empty then subtract one from its contents and go to instruction k, otherwise proceed to instruction l.

$Halt$: Stop the machine. This additional instruction can only be assigned to the final label h.

In their *deterministic variant*, such n-register machines can be used to compute any partial recursive function $f : \mathbf{N}_0^\alpha \to \mathbf{N}_0^\beta$; starting with $(n_1, ..., n_\alpha) \in \mathbf{N}_0^\alpha$ 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. If the final label cannot be reached, $f(n_1, ..., n_\alpha)$ remains undefined.

A deterministic n-register machine can also analyse an input $(n_1, ..., n_\alpha) \in \mathbf{N}_0^\alpha$ in registers 1 to α, which is recognized if the register machine finally stops by the halt instruction with all its registers being empty. If the machine does not halt, the analysis was not successful.

In their *non-deterministic variant*, n-register machines can compute any recursively enumerable set of non-negative integers (or of vectors of non-negative integers). Starting with all registers being empty, we consider a computation of the n-register machine to be successful, if it halts with the result being contained in the first (β) register(s) and with all other registers being empty.

The results proved in [5] (based on the results established in [11]) as well as in [7] and [8] immediately lead us to the following results which differ from the original results mainly by the fact that the result of a computation is stored in registers that must not be decremented:

Proposition 1 *For any partial recursive function $f : \mathbf{N}_0^\alpha \to \mathbf{N}_0^\beta$ there exists a deterministic $(\alpha + 2 + \beta)$-register machine M computing f in such a way that, when starting with $(n_1, ..., n_\alpha) \in \mathbf{N}_0^\alpha$ 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 $\alpha + 3$ to $\alpha + 2 + \beta$ 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.*

The following two corollaries are immediate consequences of the preceding proposition (by taking $\alpha = 0$ and $\beta = 0$, respectively):

Corollary 2 *For any recursively enumerable set $L \subseteq \mathbf{N}_0^\beta$ of vectors of non-negative integers there exists a non-deterministic $(\beta + 2)$-register machine M generating L in such a way that, when starting with all registers 1 to $\beta + 2$ being empty, M non-deterministically computes and halts with n_i in registers i, $3 \leq i \leq \beta + 2$, and registers 1 and 2 being empty if and only if $(n_1, ..., n_\beta) \in L$.*

Corollary 3 *For any recursively enumerable set $L \subseteq \mathbf{N}_0^\alpha$ of vectors of non-negative integers there exists a deterministic $(\alpha + 2)$-register machine M accepting L in such a way that M halts with all registers being empty if and only if M starts with some $(n_1, ..., n_\alpha) \in L$ in registers 1 to α and the registers $\alpha + 1$ to $\alpha + 2$ being empty.*

For dealing with strings, we introduce the following (quite restricted) model of an n-counter automaton, which can be seen as an n-register machine having an additional input tape for the input string to be analysed; we assume this input tape to contain the letters of the input string in the correct order followed by an arbitrary (infinite) number of blank symbols B:

An *n-counter automaton* is a construct

$$M = (n, \Sigma_B, P, i, h)$$

where

- n is the number of registers,
- Σ is the input alphabet and B is a special symbol not in Σ, $\Sigma = \{b_t \mid 1 \leq t \leq s\}$, $s \geq 1$,
- P is a set of labelled instructions of the form $j : (op(r), k, l)$, where $op(r)$ is an operation on register r of M, j, k, l are labels from the set $Lab(M)$ (which numbers the instructions of the program of M represented by P), or of the form $j : (read, k_1, ..., k_s, l)$, where $j, k_1, ..., k_s, l$ are labels from the set $Lab(M)$,
- i is the initial label, and
- h is the final label.

As an n-register machine, an n-counter automaton is capable of the following instructions on its registers:

$(A(r), k, l)$ Add one to the contents of register r and proceed to instruction k or to instruction l; in the deterministic variant we demand $k = l$.

$(S(r), k, l)$: If register r is not empty then subtract one from its contents and go to instruction k, otherwise proceed to instruction l.

Moreover, we again have the $Halt$-operation:

$Halt$: Stop the machine. This additional instruction can only be assigned to the final label h.

In addition, we now also have operations on the input tape which allow for reading the (letters of the) input string:

$j : (read, k_1, ..., k_s, l)$: Read the symbol under the read-only head of the input tape and move the read-only head of the input tape one position to the right; if this symbol equals b_t, $1 \leq t \leq s$, then go to instruction k_t, otherwise (if the read symbol equals B) proceed to instruction l.

Given a string $w \in \Sigma^+$, we say that M accepts w if and only if M, when started with w on its input tape and all counters being empty finally halts in the final label. Without loss of generality, we may assume that all counters are empty when M halts; moreover, we also assume that after reading the first blank symbol B on the input tape, M never accesses the input tape any more (because then only some more blank symbols would be read). The string language $L \subseteq \mathbf{\Sigma}^+$ accepted by M is the set of all strings $w \in \Sigma^+$ accepted by M.

¿From the results proved in [11] we know that two counters are sufficient to simulate the actions of a Turing machine, hence, the following result can easily be derived even for the special model of n-counter automata defined above:

Proposition 4 *For any recursively enumerable λ-free string language $L \subseteq \mathbf{\Sigma}^+$ there exists a (deterministic) 2-counter automaton M accepting L.*

2.2 The Standard Model of P Systems and Variants

The standard type of membrane systems (P systems) has been studied in many papers and several monographs; we refer to [1], [4], [12], [13], and [14] for motivation and examples. In the definition of the P system below we omit some ingredients (like priority relations on the rules) not needed in the following.

A *P system (of degree d, $d \geq 1$)* is a construct

$$\Pi = (V, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d, i_o) \,,$$

where:

(i) V is an alphabet; its elements are called *objects*;

(ii) $C \subseteq V$ is a set of *catalysts*;

(iii) μ is a membrane structure consisting of d membranes (usually labelled with i and represented by corresponding brackets $[_i$ and $]_i$, $1 \leq i \leq d$);

(iv) w_i, $1 \leq i \leq d$, are strings over V associated with the regions $1, 2, \ldots, d$ of μ; they represent multisets of objects present in the regions of μ (the multiplicity of a symbol in a region is given by the number of occurrences of this symbol in the string corresponding to that region);

(v) R_i, $1 \leq i \leq d$, are finite sets of *evolution rules* over V associated with the regions $1, 2, \ldots, d$ of μ; these evolution rules are of the forms $a \to v$ or $ca \to cv$, where c is a catalyst, a is an object from $V \setminus C$, and v is a string from $((V \setminus C) \times \{here, out, in\})^*$;

(vi) i_o is a number between 1 and d and it specifies the *output* membrane of Π.

The membrane structure and the multisets represented by w_i, $1 \leq i \leq d$, in Π constitute the *initial configuration* of the system. A transition between configurations is governed by the application of the evolution rules which is done in parallel: All objects, from all membranes, which *can be* the subject of local evolution rules *have to* evolve simultaneously.

The application of a rule $u \to v$ in a region containing a multiset M results in subtracting from M the multiset identified by u, and then in adding the multiset identified by v. The objects can eventually be transported through membranes due to targets *in* and *out* (we usually omit the target *here*). We refer to [1] and [14] for further details and examples. According to [9], the P system Π is called *catalytic,* if every evolution rule involves a catalyst.

The system continues parallel steps until there remain no applicable rules in any region of Π; then the system halts. We consider the number of objects from V contained in the output membrane i_o at the moment when the system halts as the *result* of the underlying computation of Π. The set of results of all computations possible in Π is denoted by $N(\Pi)$. The class of all sets of β-vectors $(y_1, ..., y_\beta)$ of non-negative integers computable by P systems (as the numbers of β different symbols to be found in the output membrane i_0 at the end of halting computations) of the above type with d membranes and the set of catalysts containing at most m elements is denoted by

$$N_0^\beta OP_{gen}(d, cat_m, halt).$$

We may relax the condition that the output membrane has to be an elementary membrane where all the elements found there at the end of a halting computation count for the result of this computation; instead, we may specify a set of terminal objects Σ and only count the number of the β different symbols of Σ to be found in any specified output membrane at the end of halting computations; in that way, we obtain *extended P systems* of the form

$$\Pi = (V, \Sigma, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d, i_0)$$

and the class

$$N_0^\beta EP_{gen}(d, cat_m, halt).$$

In addition to these generating membrane systems we may also consider accepting P systems where the multiset to be analysed is put into region i_0 together with w_{i_0} and accepted by a halting computation. The classes of all sets of β-vectors $(y_1, ..., y_\beta)$ of non-negative integers accepted in that way by halting computations in P systems of these types with d membranes and the set of catalysts containing at most m elements are denoted by

$$N_0^\beta OP_{acc}(d, cat_m, halt) \text{ and } N_0^\beta EP_{acc}(d, cat_m, halt).$$

In [2] accepting P systems were introduced as P automata using finite states as accepting conditions, i.e., instead of the halting condition an input is accepted if the P system reaches a configuration where the contents of (specified) membranes

coincides with the multisets given by a finite state. In more detail, for a P system as defined above a final state over V is of the form $(f_1, ..., f_d)$ where each f_i, $1 \leq i \leq d$, either is a final multiset over V or (a special symbol denoted by) Λ; then the P system accepts its input (given in i_0) by this final state if during the computation a configuration is reached such that the contents of every membrane i with $f_i \neq \Lambda$ coincides with f_i. The special symbol Λ indicates that we do not care about the contents of membrane i if $f_i = \Lambda$. Hence, a P system with final states is a construct of the form

$$\Pi = (V, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d, i_o, F) ,$$

where $V, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d, i_o$ are defined as above and F is a finite set of final states over V. The class of all sets of β-vectors $(y_1, ..., y_\beta)$ of non-negative integers accepted in P systems with d membranes and the set of catalysts containing at most m elements by computations reaching a final state is denoted by

$$N_0^\beta OP_{acc} (d, cat_m, \textit{final state}) .$$

Yet the idea of final states can also be carried over to generating P systems, i.e., a P system with final states as above can be used as a generative device, too; instead of considering the contents of the output membrane i_0 in halting computations we consider the contents of the output membrane i_0 in computations having reached a final state $(f_1, ..., f_d)$ (obviously, in general we must have $f_{i_0} = \Lambda$). Then the class of all sets of β-vectors $(y_1, ..., y_\beta)$ of non-negative integers generated in P systems with d membranes and the set of catalysts containing at most m elements by computations having reached a final state is denoted by

$$N_0^\beta OP_{gen} (d, cat_m, \textit{final state}) .$$

Considering *extended P systems with final states* of the form

$$\Pi = (V, \Sigma, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d, i_0, F) ,$$

where again we only take into account the terminal symbols in the specified membrane i_0, we obtain the corresponding classes

$$N_0^\beta EP_{acc} (d, cat_m, \textit{final state}) \text{ and } N_0^\beta EP_{gen} (d, cat_m, \textit{final state}) .$$

If in the variants of P systems defined above only catalytic rules are used, we add the superscript cat thus obtaining the classes

$$N_0^\beta OP_X^{cat} (d, cat_m, Y) , \ X \in \{gen, acc\} , \ Y \in \{halt, \textit{final state}\}$$

and

$$N_0^\beta EP_X^{cat} (d, cat_m, Y) , \ X \in \{gen, acc\} , \ Y \in \{halt, \textit{final state}\} .$$

We also consider variants of P systems for generating or accepting string languages:

A *P system (of degree d, $d \geq 1$) with external output* is a construct

$$\Pi = (V, \Sigma, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d) ,$$

where all ingredients of P except for Σ are defined as in the definition of the P system at the beginning of this subsection, Σ is the input alphabet, $\Sigma \cap C = \emptyset$; yet we omit the output membrane, and instead, we consider the sequence of symbols sent out during a halting computation as the generated string(s) (if more than one symbol is sent out in one step of the computation, every permutation of these symbols is taken as part of a generated string). The class of all languages $\subseteq \Sigma^+$ generated by P systems of the above type with d membranes and the set of catalysts containing at most m elements is denoted by

$$P_{gen} (d, cat_m, halt) .$$

A *P system (of degree d, $d \geq 1$) with external output and final states* is a construct of the form

$$\Pi = (V, \Sigma, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d, F) ,$$

where $V, \Sigma, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d$ are defined as above and F is a finite set of final states over V. We now consider the sequence of terminal symbols sent out during a computation having reached a final state as the generated string(s). Then the class of all languages $\subseteq \Sigma^+$ generated by P systems with external output and final states having d membranes and the set of catalysts containing at most m elements is denoted by

$$P_{gen} (d, cat_m, final\ \ state) .$$

We now also consider accepting P systems analysing a sequence of letters taken from the environment during a halting computation or during a computation stopping by reaching a final state:

A *P automaton (of degree d, $d \geq 1$) with final states* is a construct of the form

$$\Pi = (V, \Sigma, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d, F) ,$$

where $V, \Sigma, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d, F$ are defined as above for P systems with external output and final states, only the catalytic rules in the skin membrane may also contain the target indication *come* for terminal symbols, i.e., they are of the form $ca \rightarrow cv$, where c is a catalyst, a is an object from $V \setminus C$, and v is a string from

$$(((V \setminus C) \times \{here, out, in\}) \cup (\Sigma \times \{come\}))^* .$$

The target indication *come* (compare with the P systems introduced in [19]) for a terminal symbol b_t means that such a symbol is taken in from the environment (in some sense like reading it from an external input tape).

We now consider the sequence of terminal symbols taken in from the environment during a computation having reached a final state as the accepted string(s). Then

the class of all languages $\subseteq \Sigma^+$ accepted by P automata with final states having d membranes and the set of catalysts containing at most m elements is denoted by

$$P_{acc}\,(d, cat_m, \textit{final state})\,.$$

If we again consider halting computations instead of computations reaching a final state, we obtain a *P automaton (of degree d, $d \geq 1$)* as a construct of the form

$$\Pi = (V, \Sigma, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d)\,,$$

where $V, \Sigma, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d$ are defined as above for P automata with final states, we only omit the set of final states. Then the class of all languages $\subseteq \Sigma^+$ accepted (in halting computations) by P automata having d membranes and the set of catalysts containing at most m elements is denoted by

$$P_{acc}\,(d, cat_m, halt)\,.$$

If in the variants of P systems and P automata defined above only catalytic rules are used, we add the superscript cat thus obtaining the classes

$$P_X^{cat}\,(d, cat_m, Y)\,,\ X \in \{gen, acc\}\,,\ Y \in \{halt, \textit{final state}\}\,.$$

All the (catalytic) P systems with external output as well as the (catalytic) P automata taking their input from the environment as defined above can also be considered as devices for generating/accepting sets of (vectors of) non-negative integers by interpreting the strings as representations of these (vectors of) non-negative integers; in that way we obtain the classes

$$N_0^{\beta} IP_X\,(d, cat_m, Y)\,,\ X \in \{gen, acc\}\,,\ Y \in \{halt, \textit{final state}\}$$

and

$$N_0^{\beta} IP_X^{cat}\,(d, cat_m, Y)\,,\ X \in \{gen, acc\}\,,\ Y \in \{halt, \textit{final state}\}\,.$$

3 Universality Results

In order to prove the main results of this paper we elaborate a more general result using Proposition 1 that any partial recursive function $f : \mathbf{N}_0^{\alpha} \to \mathbf{N}_0^{\beta}$ can be computed by a P system (halting or with final states) with only two membranes and with only $\alpha + 2$ catalysts.

3.1 P Systems for Partial Recursive Functions

For a register machine M with m registers, $m \geq 1$, let P be the program for M with n instructions i_1, i_2, ..., i_n computing f. Informally, each register a is represented by objects o_a playing the rôles of counter elements. The value of register a at each moment corresponds to the number of symbols o_a in the system.

There are also special objects p_j, $1 \leq j \leq n$; they play the rôle of program labels and their marked variants guide the simulation of the instruction labelled by p_j within the P system. The presence of the marked variants $p_j^{\langle h,1\rangle}$, $1 \leq h \leq m$, of the object p_j - for each catalyst there has to be such a marked variant to keep it busy - starts the sequence of operations corresponding to the instruction j. For every register not representing an output value (where according to the result stated in Proposition 1 conditional decrementing may be necessary), in contrast to the proofs given in [18] and then in [6] we now need only one catalyst, because we use the concept of "paired catalysts": Together with the catalyst c_a associated with register a we also associate ("pair") another catalyst (we shall take $c_{a\ominus_m 1}$) which together with c_a will do the correct simulation of an instruction $j : (S(a), k, l) \in P$ in four steps; the remaining catalysts $c_{a\ominus_m h}$ with $2 \leq h < m$ are occupied by the marked variants of p_j, $p_j^{\langle h,l\rangle}$, $1 \leq l \leq 4$, during these four steps, and the $p_j^{\langle h,4\rangle}$ are eliminated in the fourth step, before in the next step the new multiset $p_k^{\langle 1,1\rangle}...p_k^{\langle m,1\rangle}$ or $p_l^{\langle 1,1\rangle}...p_l^{\langle m,1\rangle}$ of (marked) program labels appears. The simulation of an instruction $j : (A(a), k, k) \in P$ needs only one step. Finally, if the multiset $p_n^{\langle 1,1\rangle}...p_n^{\langle m,1\rangle}$ representing the final label n appears, these objects are also eliminated in one step, whereafter the computation halts if and only if it has been successful, i.e., no trap symbol # is still present (after having been generated during the simulation of some subtract-instruction).

Theorem 5 *For each partial recursive function $f : \mathbf{N}_0^\alpha \to \mathbf{N}_0^\beta$ there is a P system*

$$\Pi = (V, C, [_1[_2]_2]_1, w, \lambda, R, \emptyset, 2)$$

with at most $\alpha + 2$ catalysts and with the objects $o_a \in V$ satisfying the following conditions: For any arbitrary $(x_1, ..., x_\alpha) \in \mathbf{N}_0^\alpha$, denote

$$\Pi_{(x_1,...,x_\alpha)} = (V, C, [_1[_2]_2]_1, wo_1^{x_1}...o_\alpha^{x_\alpha}, \lambda, R, \emptyset, 2) .$$

The system $\Pi_{(x_1,...,x_\alpha)}$ can halt if and only if $f(x_1, ..., x_\alpha)$ is defined, and if it halts, then in the skin membrane only the catalysts remain and in the output membrane 2 only terminal symbols $o_{\alpha+3}$ to $o_{\alpha+2+\beta}$ appear in such a way that

$$N\left(\Pi_{(x_1,...,x_\alpha)}\right) = \{f(x_1, ..., x_\alpha)\} .$$

Proof. Consider a (deterministic) register machine M as defined above with m' registers, the last β registers being special output registers which are never decremented. (From the result stated in Proposition 1 we know that $m' = \alpha + 2 + \beta$ is sufficient). Now let $m = m' - \beta$ and let P be a program which computes the function f such that the initial instruction has the label 1 and the halting instruction has the label n. 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 $m + 1$ to m'. 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

$$\Pi = (V, C, [_1 [_2]_2]_1, w, \lambda, R, \emptyset, 2)$$

where

$$\begin{aligned}
V &= \{\#\} \cup \{c_i, c_i', c_i'' \mid 1 \le i \le m\} \cup \{o_k \mid 1 \le k \le m'\} \cup \\
&\left\{p_n^{\langle h,1\rangle} \mid 1 \le h \le m\right\} \cup \left\{p_j^{\langle h,1\rangle} \mid 1 \le h \le m,\ j : (A(a), k, k) \in P\right\} \cup \\
&\left\{p_j^{\langle h,1\rangle} \mid 1 \le h \le m,\ j : (S(a), k, l) \in P\right\} \cup \\
&\left\{p_j^{\langle h,l\rangle} \mid 2 \le h < m,\ 1 \le l \le 4,\ j : (S(a), k, l) \in P\right\} \cup \\
&\left\{p_j', p_j'', \bar{p}_j, \bar{p}_j', \bar{p}_j'', \hat{p}_j, \hat{p}_j', \hat{p}_j'' \mid j : (S(a), k, l) \in P\right\}, \\
C &= \{c_i \mid 1 \le i \le m\}, \\
w &= c_1 \ldots c_m p_1^{\langle 1,1\rangle} \ldots p_1^{\langle m,1\rangle}, \\
R &= \left\{x \to \# \mid x \in V \setminus \left(C \cup \{o_k \mid 1 \le k \le m'\} \cup \left\{\bar{p}_j', \hat{p}_j' \mid j : (S(a), k, l) \in P\right\}\right)\right\} \cup \\
&\left\{c_{m\ominus_m h} p_n^{\langle h,1\rangle} \to c_{m\ominus_m h} \mid 1 \le h \le m\right\} \cup \\
&\left\{c_{m\ominus_m h} p_j^{\langle h,1\rangle} \to c_{m\ominus_m h} \mid 1 \le h < m,\ 1 \le a \le m',\right. \\
&\left. j : (A(a), k, k) \in P\right\} \cup \\
&\left\{c_m p_j^{\langle m,1\rangle} \to c_m p_k^{\langle 1,1\rangle} \ldots p_k^{\langle m,1\rangle} o_a \mid 1 \le a \le m,\ j : (A(a), k, k) \in P\right\} \cup \\
&\left\{c_m p_j^{\langle m,1\rangle} \to c_m p_k^{\langle 1,1\rangle} \ldots p_k^{\langle m,1\rangle} (o_a, in) \mid m < a \le m',\right. \\
&\left. j : (A(a), k, k) \in P\right\} \cup \\
&\left\{c_{a\ominus_m h} p_j^{\langle h,l\rangle} \to c_{a\ominus_m h} p_j^{\langle h,l+1\rangle} \mid 2 \le h < m,\ 1 \le a \le m,\right. \\
&\left. 1 \le l \le 3,\ j : (S(a), k, l) \in P\right\} \cup \\
&\left\{c_{a\ominus_m h} p_j^{\langle h,4\rangle} \to c_{a\ominus_m h} \mid 2 \le h < m,\ 1 \le a \le m,\ j : (S(a), k, l) \in P\right\} \cup \\
&\left\{c_a p_j^{\langle m,1\rangle} \to c_a \hat{p}_j \hat{p}_j', c_a p_j^{\langle m,1\rangle} \to c_a \bar{p}_j \bar{p}_j' \bar{p}_j'',\right. \\
&c_a o_a \to c_a c_a', c_a c_a' \to c_a c_a'', c_{a\ominus_m 1} c_a'' \to c_{a\ominus_m 1}, \\
&c_a \hat{p}_j' \to c_a \#, c_{a\ominus_m 1} \hat{p}_j' \to c_{a\ominus_m 1} \hat{p}_j'', c_a \hat{p}_j'' \to c_a p_k^{\langle 1,1\rangle} \ldots p_k^{\langle m,1\rangle}, \\
&c_a \bar{p}_j \to c_a, c_{a\ominus_m 1} \bar{p}_j'' \to c_{a\ominus_m 1} p_j'', c_{a\ominus_m 1} p_j'' \to c_{a\ominus_m 1} p_j', \\
&\left. c_a p_j' \to c_a p_l^{\langle 1,1\rangle} \ldots p_l^{\langle m,1\rangle} \mid 1 \le a \le m,\ j : (S(a), k, l) \in P\right\} \cup \\
&\left\{c_{a\ominus_m 1} y \to c_{a\ominus_m 1} \mid y \in \left\{p_j^{\langle 1,1\rangle}, \hat{p}_j, \bar{p}_j'\right\}, 1 \le a \le m,\right. \\
&\left. j : (S(a), k, l) \in P\right\}.
\end{aligned}$$

Then for an arbitrary $(x_1, ..., x_\alpha) \in \mathbf{N}_0^\alpha$ the axiom of the corresponding system $\Pi_{(x_1,\ldots,x_\alpha)}$ is

$$c_1 \ldots c_m p_1^{\langle 1,1\rangle} \ldots p_1^{\langle m,1\rangle} o_1^{x_1} \ldots o_\alpha^{x_\alpha}.$$

The contents of register a, $1 \leq a \leq m$, is represented by the sum of the number of symbols o_a and conditional decrementing actions on this register are guarded by the pair of catalysts c_a and $c_{a\ominus_m 1}$.

The set of rules R depends on the instructions of P; the halting instruction as well as each add-instruction is simulated in one step, whereas each subtract-instruction is simulated in four steps; in more detail, the simulation works as follows:

1. Every simulaion of a rule starts with the program labels $p_1^{\langle 1,1\rangle}, ..., p_1^{\langle m,1\rangle}$. The halting instruction eliminates the final labels $p_n^{\langle 1,1\rangle}, ..., p_n^{\langle m,1\rangle}$ by using the rules $c_{m\ominus_m h} p_n^{\langle h,1\rangle} \rightarrow c_{m\ominus_m h}$, $1 \leq h \leq m$; if the computation has been successful, then only the catalysts remain in the skin membrane, whereas the result of the computation, i.e., the number of symbols o_{m+1} to $o_{m+\beta}$, can be found in the output membrane 2.

2. Each add-instruction $j : (A(a), k, k) \in P$, $1 \leq a \leq m$ ($m < a \leq m'$, respectively) is simulated in one step by using the catalytic rules $c_{m\ominus_m h} p_j^{\langle h,1\rangle} \rightarrow c_{m\ominus_m h}$, $1 \leq h < m$, as well as $c_m p_j^{\langle m,1\rangle} \rightarrow c_m p_k^{\langle 1,1\rangle} ... p_k^{\langle m,1\rangle} o_a$ ($c_m p_j^{\langle m,1\rangle} \rightarrow c_m p_k^{\langle 1,1\rangle} ... p_k^{\langle m,1\rangle} (o_a, in)$, respectively, i.e., if a is the index of a register representing a component of the result vector of the computation, then the symbol o_a is immediately moved to the output membrane 2). Observe that by definition $a \ominus_m m = a$ for all a with $1 \leq a \leq m$.

3. Each subtract-instruction $j : (S(a), k, l) \in P$ is simulated in four steps. We have to distinguish between two cases depending on the contents of register a; in both cases the catalysts $c_{a\ominus_m h}$, $2 \leq h < m$, are busy with the objects $p_j^{\langle h,l\rangle}$, $1 \leq l \leq 4$; the objects $p_j^{\langle h,4\rangle}$ finally are eliminated in the fourth step. The main part of the simulation is accomplished by the catalyst c_a and its "paired companion" $c_{a\ominus_m 1}$:

   (a) We non-deterministically assume that the contents of register a is not empty; we start with the rules $c_a p_j^{\langle m,1\rangle} \rightarrow c_a \hat{p}_j \hat{p}'_j$ and $c_{a\ominus_m 1} p_j^{\langle 1,1\rangle} \rightarrow c_{a\ominus_m 1}$. In the second step, the number of symbols o_a is decremented by using the rule $c_a o_a \rightarrow c_a c'_a$; if in contrast to our choice, no such symbol o_a is present (i.e., the contents of the register represented by the number of symbols o_a is empty), then by the enforced application of the rule $c_a \hat{p}'_j \rightarrow c_a \#$ the trap symbol $\#$ is introduced, which causes a non-halting computation due to the rule $\# \rightarrow \#$. If $\hat{p}'_j$ could wait until being used in the third step by the rule $c_{a\ominus_m 1} \hat{p}'_j \rightarrow c_{a\ominus_m 1} \hat{p}''_j$, then the simulation will be successful: In the second step, $c_{a\ominus_m 1}$ is used in the rule $c_{a\ominus_m 1} \hat{p}_j \rightarrow c_{a\ominus_m 1}$, and in the third step c_a is used in the rule $c_a c'_a \rightarrow c_a c''_a$. We finish with the application of the rules $c_a \hat{p}''_j \rightarrow c_a p_k^{\langle 1,1\rangle} ... p_k^{\langle m,1\rangle}$ and $c_{a\ominus_m 1} c''_a \rightarrow c_{a\ominus_m 1}$.

   (b) For the other case, we non-deterministically assume that the contents of register a is empty; we start with the two rules $c_a p_j^{\langle m,1\rangle} \rightarrow c_a \bar{p}_j \bar{p}'_j \bar{p}''_j$

and $c_{a\ominus_m 1}p_j^{\langle 1,1\rangle} \to c_{a\ominus_m 1}$. In the second step, we are forced to use the two rules $c_a\bar{p}_j \to c_a$ and $c_{a\ominus_m 1}\bar{p}_j'' \to c_{a\ominus_m 1}p_j''$ in order not to introduce the trap symbol #. In the third step, we only use $c_{a\ominus_m 1}p_j'' \to c_{a\ominus_m 1}p_j'$ and finish with applying the two rules $c_a p_j' \to c_a p_l^{\langle 1,1\rangle}...p_l^{\langle m,1\rangle}$ and $c_{a\ominus_m 1}\bar{p}_j' \to c_{a\ominus_m 1}$ in the fourth step. In the third step the catalyst c_a is not used if our non-deterministic choice has been correct, i.e., if there is no symbol o_a present in the skin membrane; otherwise, the rule $c_a o_a \to c_a c_a'$ has to be applied in the third step, but in this case both c_a' and p_j' would need the catalyst c_a in the fourth step of the simulation in order not to be sent to the trap symbol #.

Any other behavior of the system as the one described above for the correct simulation of the instructions of P by the rules in R leads to the appearance of the trap object # within the system, hence, the system never halts.

It follows from the description given above that after each simulation of an instruction the number of objects o_a equals the contents of register a, $1 \leq a \leq m'$. Hence, after having simulated the instruction $Halt$ and halting the system, the number of symbols o_{m+1} to $o_{m+\beta}$ in the output membrane 2 equals the output of the program P. The only other objects remaining within the system are the m catalysts in the skin membrane; according to the result about register machines stated in Proposition 1, $m = \alpha + 2$ and therefore $\alpha + 2$ catalysts are enough. □

For P systems with final states, we can immediately take over the construction given in the preceding proof:

Corollary 6 *For each partial recursive function $f : \mathbf{N}_0^\alpha \to \mathbf{N}_0^\beta$ there is a P system with final states*

$$\Pi^F = (V, C, [_1[_2]_2]_1, w, \lambda, R, \emptyset, 2, F)$$

with at most $\alpha + 2$ catalysts and with the objects $o_a \in V$ satisfying the following conditions: For any arbitrary $(x_1, ..., x_\alpha) \in \mathbf{N}_0^\alpha$, denote

$$\Pi^F_{(x_1,...,x_\alpha)} = (V, C, [_1[_2]_2]_1, wo_1^{x_1}...o_\alpha^{x_\alpha}, \lambda, R, \emptyset, 2, F)\,.$$

The system $\Pi^F_{(x_1,...,x_\alpha)}$ reaches a final state if and only if $f(x_1, ..., x_\alpha)$ is defined, and in the final state the output membrane 2 contains only terminal symbols o_{m+1} to $o_{m+\beta}$ in such a way that

$$N\left(\Pi^F_{(x_1,...,x_\alpha)}\right) = \{f(x_1, ..., x_\alpha)\}\,.$$

Proof. The only difference to the P system constructed in Theorem 5 is that we have to define the final state for successful computations, which simply is the contents of the skin membrane at the end of a halting computation, i.e., $c_1 \ldots c_m$. Hence, taking $F = \{(c_1 \ldots c_m, \Lambda)\}$ we obtain the P system with final states Π' is $(\Pi, \{(c_1 \ldots c_m, \Lambda)\})$, where Π is the P system constructed in the proof of Theorem 5.

□

In catalytic systems we only need one more catalyst for the rules handling the trap symbol #:

Corollary 7 *For each partial recursive function $f : \mathbf{N}_0^\alpha \to \mathbf{N}_0^\beta$ there is*

1. *a halting catalytic P system*

$$\Pi^{cH} = (V \cup \{c_0\}, C \cup \{c_0\}, [_1[_2]_2]_1, w, \lambda, R_C, \emptyset, 2),$$

2. *a catalytic P system with final states*

$$\Pi^{cF} = (V \cup \{c_0\}, C \cup \{c_0\}, [_1[_2]_2]_1, w, \lambda, R_C, \emptyset, 2, F),$$

respectively, with at most $\alpha + 3$ catalysts and with the objects $o_a \in V$ satisfying the following conditions: For any arbitrary $(x_1, ..., x_\alpha) \in \mathbf{N}_0^\alpha$, denote

1. $\Pi^{cH}_{(x_1,...,x_\alpha)} = (V \cup \{c_0\}, C \cup \{c_0\}, [_1[_2]_2]_1, wo_1^{x_1}...o_\alpha^{x_\alpha}, \lambda, R_C, \emptyset, 2)$ *and*

2. $\Pi^{cF}_{(x_1,...,x_\alpha)} = (V \cup \{c_0\}, C \cup \{c_0\}, [_1[_2]_2]_1, wo_1^{x_1}...o_\alpha^{x_\alpha}, \lambda, R_C, \emptyset, 2, F),$

respectively. The system

1. $\Pi^{cH}_{(x_1,...,x_\alpha)}$ *halts,*

2. $\Pi^{cF}_{(x_1,...,x_\alpha)}$ *reaches a final state,*

respectively, if and only if $f(x_1, ..., x_\alpha)$ is defined, and

1. *in the halting computation or*

2. *in the final state*

respectively, in the skin membrane only the catalysts remain and the output membrane 2 contains only terminal symbols o_{m+1} to $o_{m+\beta}$ in such a way that

$$N\left(\Pi^{cH}_{(x_1,...,x_\alpha)}\right) = N\left(\Pi^{cF}_{(x_1,...,x_\alpha)}\right) = \{f(x_1, ..., x_\alpha)\}.$$

Proof. The rules in R_C are obtained from the rules in R constructed in the proof of Theorem 5 by just replacing the rules in

$$\left\{x \to \# \mid x \in V \setminus \left(C \cup \left\{\bar{p}'_j, \hat{p}'_j \mid j : (S(a), k, l) \in P\right\} \cup \{o_k \mid 1 \le k \le m'\}\right)\right\}$$

with the rules in

$$\left\{c_0 x \to c_0 \# \mid x \in V \setminus \left(C \cup \left\{\bar{p}'_j, \hat{p}'_j \mid j : (S(a), k, l) \in P\right\} \cup \{o_k \mid 1 \le k \le m'\}\right)\right\}$$

using the additional catalyst c_0. □

In P systems with external output and in extended P systems we do not need the additional output membrane, i.e., $\alpha + 2$ catalysts ($\alpha + 3$ catalysts in catalytic systems) in the skin membrane are sufficient:

Corollary 8 *For each partial recursive function $f : \mathbf{N}_0^\alpha \to \mathbf{N}_0^\beta$ there is*

1. *a (halting) P system with external output*

$$\Pi^O = (V, \Sigma, C, [_1]_1, w, R_O)$$

with at most $\alpha + 2$ catalysts,

2. *a P system with external output and final states*

$$\Pi^{OF} = (V, \Sigma, C, [_1]_1, w, R_O, \{c_1 \dots c_m\})$$

with at most $\alpha + 2$ catalysts,

3. *a catalytic (halting) P system with external output*

$$\Pi^{cO} = (V \cup \{c_0\}, \Sigma, C \cup \{c_0\}, [_1]_1, w, R_O)$$

with at most $\alpha + 3$ catalysts,

4. *a catalytic P system with external output and final states*

$$\Pi^{cOF} = (V \cup \{c_0\}, \Sigma, C \cup \{c_0\}, [_1]_1, w, R_O, \{c_0 c_1 \dots c_m\})$$

with at most $\alpha + 3$ catalysts,

respectively, with $\Sigma = \{o_k \mid m+1 \leq k \leq m+\beta\}$ and the objects $o_a \in V$ satisfying the following conditions: For any arbitrary $(x_1, ..., x_\alpha) \in \mathbf{N}_0^\alpha$, denote the corresponding P system by

1. $\Pi^O_{(x_1,\dots,x_\alpha)} = (V, \Sigma, C, [_1]_1, wo_1^{x_1} ... o_\alpha^{x_\alpha}, R_O)$,
2. $\Pi^{OF}_{(x_1,\dots,x_\alpha)} = (V, \Sigma, C, [_1]_1, wo_1^{x_1} ... o_\alpha^{x_\alpha}, R_O, \{c_1 \dots c_m\})$,
3. $\Pi^{cO}_{(x_1,\dots,x_\alpha)} = (V \cup \{c_0\}, \Sigma, C \cup \{c_0\}, [_1]_1, wo_1^{x_1} ... o_\alpha^{x_\alpha}, R_O)$,
4. $\Pi^{cOF}_{(x_1,\dots,x_\alpha)} = (V \cup \{c_0\}, \Sigma, C \cup \{c_0\}, [_1]_1, wo_1^{x_1} ... o_\alpha^{x_\alpha}, R_O, \{c_0 c_1 \dots c_m\})$,

respectively.

1. $\Pi^O_{(x_1,\dots,x_\alpha)}$ *halts,*
2. $\Pi^{cO}_{(x_1,\dots,x_\alpha)}$ *halts,*
3. $\Pi^{OF}_{(x_1,\dots,x_\alpha)}$ *reaches the final state $c_1 \dots c_m$,*
4. $\Pi^{cOF}_{(x_1,\dots,x_\alpha)}$ *reaches the final state $c_0 c_1 \dots c_m$,*

respectively, if and only if $f(x_1,...,x_\alpha)$ is defined, and after halting the computation or after having reached the final state, respectively, in the skin membrane only the catalysts remain and the terminal symbols o_{m+1} to $o_{m+\beta}$ are sent out during the computation in such a way that

$$N\left(\Pi^G_{(x_1,...,x_\alpha)}\right) = \{f(x_1,...,x_\alpha)\} \text{ for } G \in \{O, OF, cO, cOF\}.$$

Proof. The rules in R_O are obtained from the rules in R constructed in the proof of Theorem 5 (and Corollary 6) as well as from the rules in R_C constructed in the proof of Corollary 7 by just replacing each occurrence of (o_a, in) by (o_a, out). □

Corollary 9 *For each partial recursive function $f : \mathbf{N}_0^\alpha \to \mathbf{N}_0^\beta$ there is*

1. *an extended (halting) P system*

$$\Pi^E = (V, \Sigma, C, [_1]_1, w, R_E)$$

   with at most $\alpha + 2$ catalysts,

2. *an extended P system with final states*

$$\Pi^{EF} = (V, \Sigma, C, [_1]_1, w, R_E, \{c_1 \dots c_m\})$$

   with at most $\alpha + 2$ catalysts,

3. *a catalytic extended (halting) P system*

$$\Pi^{cE} = (V \cup \{c_0\}, \Sigma, C \cup \{c_0\}, [_1]_1, w, R_E)$$

   with at most $\alpha + 3$ catalysts,

4. *a catalytic extended P system with final states*

$$\Pi^{cEF} = (V \cup \{c_0\}, \Sigma, C \cup \{c_0\}, [_1]_1, w, R_E, \{c_0 c_1 \dots c_m\})$$

   with at most $\alpha + 3$ catalysts,

respectively, with $\Sigma = \{o_k \mid m+1 \le k \le m+\beta\}$ and the objects $o_a \in V$ satisfying the following conditions: For any arbitrary $(x_1,...,x_\alpha) \in \mathbf{N}_0^\alpha$, denote the corresponding P system by

1. $\Pi^E_{(x_1,...,x_\alpha)} = (V, \Sigma, C, [_1]_1, wo_1^{x_1}...o_\alpha^{x_\alpha}, R_E)$,
2. $\Pi^{EF}_{(x_1,...,x_\alpha)} = (V, \Sigma, C, [_1]_1, wo_1^{x_1}...o_\alpha^{x_\alpha}, R_E, \{c_1 \dots c_m\})$,
3. $\Pi^{cE}_{(x_1,...,x_\alpha)} = (V \cup \{c_0\}, \Sigma, C \cup \{c_0\}, [_1]_1, wo_1^{x_1}...o_\alpha^{x_\alpha}, R_E)$,
4. $\Pi^{cEF}_{(x_1,...,x_\alpha)} = (V \cup \{c_0\}, \Sigma, C \cup \{c_0\}, [_1]_1, wo_1^{x_1}...o_\alpha^{x_\alpha}, R_E, \{c_0 c_1 \dots c_m\})$,

respectively.

1. $\Pi^{E}_{(x_1,...,x_\alpha)}$ *halts,*
2. $\Pi^{cE}_{(x_1,...,x_\alpha)}$ *halts,*
3. $\Pi^{EF}_{(x_1,...,x_\alpha)}$ *reaches the final state* $c_1 \dots c_m$,
4. $\Pi^{cEF}_{(x_1,...,x_\alpha)}$ *reaches the final state* $c_0 c_1 \dots c_m$,

respectively, if and only if $f(x_1, ..., x_\alpha)$ *is defined, and after halting the computation or after having reached the final state, respectively, in the skin membrane only the catalysts and the terminal symbols* o_{m+1} *to* $o_{m+\beta}$ *remain in such a way that*

$$N\left(\Pi^{G}_{(x_1,...,x_\alpha)}\right) = \{f(x_1, ..., x_\alpha)\} \text{ for } G \in \{E, EF, cE, cEF\}.$$

Proof. The rules in R_E are obtained from the rules in R constructed in the proof of Theorem 5 (and Corollary 6) as well as from the rules in R_C constructed in the proof of Corollary 7 by just replacing each occurrence of (o_a, in) by o_a (which in fact means $(o_a, here)$). □

3.2 Generating P Systems

The following corollaries are immediate consequences of Theorem 5 as well as Corollaries 6, 7, 8 and 9 by taking $\alpha = 0$.

Corollary 10 $N_0^\beta OP_{gen}(d, cat_2, halt) = N_0^\beta RE$ *for every* $d \geq 2$.

Proof. We only have to prove the inclusion $N_0^\beta RE \subseteq N_0^\beta OP_{gen}(2, cat_2, halt)$. In the same way as in the proof of Theorem 5 the P system Π was constructed in order to simulate the (deterministic) register machine from Proposition 1, we now construct a P system Π' which simulates the non-deterministic register machine from Corollary 2 and in that way non-deterministically generates a representation of any vector from the given language L in $N_0^\beta RE$ by the corresponding numbers of symbols o_3 to $o_{2+\beta}$. Hence, let us define

$$\Pi' = (V, C, [_1[_2]_2]_1, w, \lambda, R', \emptyset, 2)$$

where R' is constructed in a similar way as R in the proof of Theorem 5, except that now in the non-deterministic case we have add-instructions of the form $j : (A(a), k, l)$ for some a, k, l with $a \in \{1, 2\}$ and $1 \leq k, l \leq n$; for their simulation we now not only need the rule $c_m p_j^{\langle m,1\rangle} \to c_m p_k^{\langle 1,1\rangle} ... p_k^{\langle m,1\rangle} o_a$, but also $c_m p_j^{\langle m,1\rangle} \to c_m p_l^{\langle 1,1\rangle} ... p_l^{\langle m,1\rangle} o_a$ in R'. Obviously, $N(\Pi') = L$. By the given construction, we only need 2 catalysts!

As the result is interesting of its own, we completely specify Π'; as only two catalysts are needed, we can use a less complex notation, because these two catalysts form the only pair used in the simulation of any subtract-instruction, i.e., we do

not need the objects $p_j^{\langle h,l\rangle}$, $2 \leq h < m$, $1 \leq l \leq 4$, for the subtract-instructions $j : (S(a), k, l) \in P$. Using p_j and $\tilde{p}_j$ instead of $p_j^{\langle m,1\rangle}$ and $p_j^{\langle 1,1\rangle}$, we obtain the following P system Π':

$$
\begin{aligned}
V &= \{\#\} \cup \{c_1, c_1', c_1'', c_2, c_2', c_2''\} \cup \{o_k \mid 1 \leq k \leq m'\} \cup \\
&\quad \left\{p_j, \tilde{p}_j, p_j'', \bar{p}_j, \bar{p}_j', \bar{p}_j'', \hat{p}_j, \hat{p}_j', \hat{p}_j'' \mid j : (S(a), k, l) \in P\right\} \cup \\
&\quad \{p_j, \tilde{p}_j \mid j : (A(a), k, l) \in P\}, \\
C &= \{c_1, c_2\}, \\
w &= c_1 c_2 p_1 \tilde{p}_1, \\
R' &= \left\{x \to \# \mid x \in \left\{p_j, \tilde{p}_j, p_j'', \bar{p}_j, \bar{p}_j'', \hat{p}_j, \hat{p}_j'' \mid j : (S(a), k, l) \in P\right\}\right\} \cup \\
&\quad \{\# \to \#\} \cup \{c_1 p_n \to c_1, c_2 \tilde{p}_n \to c_2\} \cup \\
&\quad \{c_1 \tilde{p}_j \to c_1 \mid j : (A(a), k, l) \in P\} \cup \\
&\quad \{c_2 p_j \to c_2 p_k \tilde{p}_k o_a, c_2 p_j \to c_2 p_l \tilde{p}_l o_a \mid \\
&\quad\ a \in \{1, 2\},\ j : (A(a), k, l) \in P\} \cup \\
&\quad \{c_2 p_j \to c_2 p_k \tilde{p}_k (o_a, in) \mid 2 < a \leq m',\ j : (A(a), k, k) \in P\} \cup \\
&\quad \left\{c_a p_j \to c_a \hat{p}_j \hat{p}_j', c_a p_j \to c_a \bar{p}_j \bar{p}_j' \bar{p}_j'', \right. \\
&\quad\ c_a o_a \to c_a c_a', c_a c_a' \to c_a c_a'', c_{3-a} c_a'' \to c_{3-a}, \\
&\quad\ c_a \hat{p}_j' \to c_a \#, c_{3-a} \hat{p}_j' \to c_{3-a} \hat{p}_j'', c_a \hat{p}_j'' \to c_a p_k \tilde{p}_k, \\
&\quad\ c_a \bar{p}_j \to c_a, c_{3-a} \bar{p}_j'' \to c_{3-a} p_j'', c_{3-a} p_j'' \to c_{3-a} p_j', \\
&\quad\ \left. c_a p_j' \to c_a p_l \tilde{p}_l \mid a \in \{1, 2\},\ j : (S(a), k, l) \in P\right\} \cup \\
&\quad \left\{c_{3-a} y \to c_{3-a} \mid y \in \left\{\tilde{p}_j, \hat{p}_j, \bar{p}_j'\right\}, a \in \{1, 2\}, \right. \\
&\quad\ \left. j : (S(a), k, l) \in P\right\}.
\end{aligned}
$$

The following table shows how a subtract-instruction $j : (S(a), k, l) \in P$ is simulated depending on the contents of register a; the rules in the brackets ($\left\langle c_a \hat{p}_j' \to c_a \# \right\rangle$ as well as $\langle c_a o_a \to c_a c_a' \rangle$ and $\langle c_{3-a} o_{3-a} \to c_{3-a} c_{3-a}' \rangle$) are those which should not be applied at that stage of the simulation; their application (only the application of the rule $\left\langle c_a \hat{p}_j' \to c_a \# \right\rangle$ may even be forced due to maximal paralellism) leads to the introduction of the failure symbol # (directly or one step later) and therefore to a non-halting computation.

simulation of the subtract-instruction $j : (S(a), k, l)$ if	
the contents of register a is not empty	the contents of register a is empty
$c_a p_j \to c_a \hat{p}_j \hat{p}'_j$ $c_{3-a} \tilde{p}_j \to c_{3-a}$	$c_a p_j \to c_a \bar{p}_j \bar{p}'_j \bar{p}''_j$ $c_{3-a} \tilde{p}_j \to c_{3-a}$
$c_a o_a \to c_a c'_a$ $\left\langle c_a \hat{p}'_j \to c_a \# \right\rangle$ $c_{3-a} \hat{p}_j \to c_{3-a}$	$c_a \bar{p}_j \to c_a$ $c_{3-a} \bar{p}''_j \to c_{3-a} p''_j$
$c_a c'_a \to c_a c''_a$ $c_{3-a} \hat{p}'_j \to c_{3-a} \hat{p}''_j$ $\langle c_{3-a} o_{3-a} \to c_{3-a} c'_{3-a} \rangle$	$\langle c_a o_a \to c_a c'_a \rangle$ $c_{3-a} p''_j \to c_{3-a} p'_j$
$c_a \hat{p}''_j \to c_a p_k \tilde{p}_k$ $c_{3-a} c''_a \to c_{3-a}$	$c_a p'_j \to c_a p_l \tilde{p}_l$ $c_{3-a} \bar{p}'_j \to c_{3-a}$ $\langle c_{3-a} o_{3-a} \to c_{3-a} c'_{3-a} \rangle$

We should like to mention that at any time c_a can be used in the rule $c_a o_a \to c_a c'_a$, but carried out at the wrong time, the application of this rule will immediately cause the introduction of the trap symbol #. □

Corollary 11 *$N_0^\beta OP_{gen}(d, cat_2, final\ state) = N_0^\beta RE$ for every $d \geq 2$.*

Proof. In the same way as in the proof of Corollary 6 the P system Π^F was constructed from the P system Π constructed in the proof of Theorem 5 we now can construct the P system with final states generating a set $L \in N_0^\beta RE$ from the P system constructed in the proof of Corollary 10. □

Obviously, the results obtained so far are optimal with respect to the number of membranes in the P systems constructed in the proofs of Theorem 5 and Corollaries 6 to 11. As far as P systems generating sets from $N_0^\beta RE$ are concerned we should like to recall the fact that without priorities as well as without catalysts, too, we can generate only regular sets. Hence, only the case of such P systems with one catalyst needs further investigations.

For catalytic P systems, in [9] it was proved that with one catalyst we cannot reach universal computational power; hence, only the case of two catalysts in catalytic P systems remains for a suitable characterization, because from Corollaries 10 and 11 we immediately infer the following results (in the same way as Corollary 7 was an obvious consequence of Theorem 5 and Corollary 6):

Corollary 12 *For every $d \geq 2$, we have*

$$\begin{aligned} N_0^\beta RE &= N_0^\beta OP_{gen}^{cat}(d, cat_3, halt) \\ &= N_0^\beta OP_{gen}^{cat}(d, cat_3, final\ state). \end{aligned}$$

The proofs of the following results immediately follow from preceding proofs, too (see Corollaries 8 and 9):

Corollary 13 *For every $d \geq 1$, we have*

$$\begin{aligned} N_0^\beta RE &= N_0^\beta XP_{gen}(d, cat_2, Y) \\ &= N_0^\beta XP_{gen}^{cat}(d, cat_3, Y) \end{aligned}$$

for every $X \in \{O, I\}$ and $Y \in \{halt, final\ state\}$.

3.3 Accepting P Systems / P Automata

The following corollaries are immediate consequences of Theorem 5 as well as Corollaries 6, 7, 8 and 9 by taking $\beta = 0$. Although for P automata we now have the minimal number of only one membrane, the number of catalysts depends on the number α of components of the vector of non-negative integers to be analysed.

Corollary 14 *For every $d \geq 1$, we have*

$$\begin{aligned} N_0^\alpha RE &= N_0^\alpha XP_{acc}(d, cat_{\alpha+2}, Y) \\ &= N_0^\alpha XP_{acc}^{cat}(d, cat_{\alpha+3}, Y) \end{aligned}$$

for every $X \in \{O, E, I\}$ and $Y \in \{halt, final\ state\}$.

Proof. We first prove the inclusion $N_0^\alpha RE \subseteq N_0^\alpha OP_{acc}(1, cat_{\alpha+2}, halt)$. In the same way as in the proof of Theorem 5 the P system there was constructed in order to simulate the (deterministic) register machine from Proposition 1, we now construct a P system which simulates the (deterministic) register machine from Corollary 3. As we have no output, we simply can omit the output membrane; moreover, we have no rules sending an object into another membrane. The rest of the construction is exactly the same as in Theorem 5.

For the remaining variants of accepting P systems and P automata we only refer to the proof ideas elaborated in the preceding proofs. □

For the simplest case of $\alpha = 1$, therefore the maximal number of catalysts needed for accepting languages from $N_0^\alpha RE$ by P systems is 3 and by catalytic P systems is 4.

3.4 P Systems Accepting/Generating Strings

Theorem 15 *For every $d \geq 1$, we have*

$$\begin{aligned} RE &= P_{gen}(d, cat_2, halt) \\ &= P_{acc}(d, cat_2, halt) \\ &= P_{gen}(d, cat_2, final\ state) \\ &= P_{acc}(d, cat_2, final\ state). \end{aligned}$$

Proof. We first prove the inclusion $RE \subseteq P_{gen}(1, cat_2, halt)$. In the same way as in the proof of Corollary 10 we simulated the operations on the two registers allowing for decrementation we now simulate the operations on the two counters of the 2-counter automaton from Proposition 4

$$M = (n, \Sigma_B, P, i, h),$$

where $\Sigma = \{b_t \mid 1 \leq t \leq s\}$, $s \geq 1$. Hence, let us define

$$\Pi = (V, \Sigma, \{c_1, c_2\}, [_1]_1, w, R, \emptyset)$$

where R is constructed in a similar way as R' in the proof of Corollary 10, except that now we also have to consider instructions of the form $j : (read, k_1, ..., k_s, l)$, which are simulated (in a non-deterministic way) by the following rules:

$$\begin{array}{l} c_1\tilde{p}_j \to c_1, \\ c_2p_j \to c_2p_{k_t}\tilde{p}_{k_t}(b_t, out), \ 1 \leq t \leq s, \\ c_2p_j \to c_2p_l\tilde{p}_l. \end{array}$$

In sum, we obtain the following P system with external output Π:

$$\begin{array}{rcl} V & = & \{\#\} \cup \{c_1, c_1', c_1'', c_2, c_2', c_2''\} \cup \{o_1, o_2\} \cup \\ & & \left\{p_j, \tilde{p}_j, p_j'', \bar{p}_j, \bar{p}_j', \bar{p}_j'', \hat{p}_j, \hat{p}_j', \hat{p}_j'' \mid j : (S(a), k, l) \in P\right\} \cup \\ & & \{p_j, \tilde{p}_j \mid j : (A(a), k, l) \in P\} \cup \Sigma \cup \{B\}, \\ C & = & \{c_1, c_2\}, \\ w & = & c_1c_2p_1\tilde{p}_1, \\ R & = & \left\{x \to \# \mid x \in V \setminus \left(C \cup \{o_1, o_2\} \cup \left\{\bar{p}_j', \hat{p}_j' \mid j : (S(a), k, l) \in P\right\}\right)\right\} \cup \\ & & \{c_1p_n \to c_1, c_2\tilde{p}_n \to c_2\} \cup \\ & & \{c_1\tilde{p}_j \to c_1 \mid j : (A(a), k, l) \in P\} \cup \\ & & \{c_2p_j \to c_2p_k\tilde{p}_ko_a, c_2p_j \to c_2p_l\tilde{p}_lo_a \mid \\ & & a \in \{1, 2\}, \ j : (A(a), k, l) \in P\} \cup \\ & & \left\{c_ap_j \to c_a\hat{p}_j\hat{p}_j', c_ap_j \to c_a\bar{p}_j\bar{p}_j'\bar{p}_j'', \right. \\ & & c_ao_a \to c_ac_a', c_ac_a' \to c_ac_a'', c_{3-a}c_a'' \to c_{3-a}, \\ & & c_a\hat{p}_j' \to c_a\#, c_{3-a}\hat{p}_j' \to c_{3-a}\hat{p}_j'', c_a\hat{p}_j'' \to c_ap_k\tilde{p}_k, \\ & & c_a\bar{p}_j \to c_a, c_{3-a}\bar{p}_j'' \to c_{3-a}p_j'', c_{3-a}p_j'' \to c_{3-a}p_j', \\ & & \left.c_ap_j' \to c_ap_l\tilde{p}_l \mid a \in \{1, 2\}, \ j : (S(a), k, l) \in P\right\} \cup \\ & & \left\{c_{3-a}y \to c_{3-a} \mid y \in \left\{\tilde{p}_j, \hat{p}_j, \bar{p}_j'\right\}, a \in \{1, 2\}, \right. \\ & & \left. j : (S(a), k, l) \in P\right\} \cup \\ & & \{c_1\tilde{p}_j \to c_1 \mid j : (read, k_1, ..., k_s, l) \in P\} \cup \\ & & \{c_2p_j \to c_2p_{k_t}\tilde{p}_{k_t}(b_t, out) \mid j : (read, k_1, ..., k_s, l) \in P, \ 1 \leq t \leq s, \} \cup \\ & & \{c_2p_j \to c_2p_l\tilde{p}_l \mid j : (read, k_1, ..., k_s, l) \in P\}. \end{array}$$

For the corresponding accepting P systems (P automata) we simply have to replace the rules

$$c_2 p_j \to c_2 p_{k_t} \tilde{p}_{k_t} \, (b_t, out)$$

by the corresponding rules

$$c_2 p_j \to c_2 p'_{j,t} \, (b_t, come) \, , \; c_2 p'_{j,t} \to c_2 p_{k_t} \tilde{p}_{k_t}, \; c_1 b_t \to c_1,$$

which proves $RE \subseteq P_{acc} \, (1, cat_2, halt)$; of course, we have to add the new symbols $p'_{j,t}$ to V.

For the corresponding P systems with external output and final states and for the corresponding P automata with final states we use the final state $c_1 c_2$, which immediately proves $RE \subseteq P_{gen} \, (1, cat_2, final \; state)$ and $RE \subseteq P_{acc} \, (1, cat_2, final \; state)$.

□

For the catalytic variants we need one more catalyst (compare with Corollary 7):

Corollary 16 *For every $d \geq 1$, we have*

$$\begin{aligned} RE &= P^{cat}_{gen} \, (d, cat_3, halt) \\ &= P^{cat}_{acc} \, (d, cat_3, halt) \\ &= P^{cat}_{gen} \, (d, cat_3, final \; state) \\ &= P^{cat}_{acc} \, (d, cat_3, final \; state) \, . \end{aligned}$$

Proof. As in the proof of Corollary 7 we replace the rules in

$$\left\{ x \to \# \mid x \in V \setminus \left(C \cup \{o_1, o_2\} \cup \left\{ \bar{p}'_j, \hat{p}'_j \mid j : (S \, (a) \, , k, l) \in P \right\} \right) \right\}$$

in the sets of rules constructed in Theorem 15 with the rules in

$$\left\{ c_0 x \to c_0 \# \mid x \in V \setminus \left(C \cup \{o_1, o_2\} \cup \left\{ \bar{p}'_j, \hat{p}'_j \mid j : (S \, (a) \, , k, l) \in P \right\} \right) \right\}$$

using the additional catalyst c_0.

□

4 Conclusion

The number of catalysts used in the P systems constructed in the proofs of this paper can be seen as a complexity measure for these systems. Only the characterization of functions computed or sets generated/accepted by the variants of P systems considered in this paper having one catalyst less remains as an interesting open question for future research; yet we conjecture that for computationally universal P systems the results obtained in this paper are already optimal not only with respect to the number of membranes, but also with respect to the number of catalysts.

In [10], the bounds for the number of catalysts and/or membranes were tried to be improved (with respect to the optimal results known before this paper) by introducing more powerful types of catalysts like so-called bi-stable catalysts and mobile catalysts. The authors showed that a P system can generate all recursively

enumerable number sets using (a) five membranes, two catalysts and one bi-stable catalyst, (b) three membranes and one mobile catalyst, (c) two membranes and two mobile catalysts. From these results, case (a) has become obsolete by the results obtained in this paper, whereas case (b) may give a chance for improving this result for mobile catalysts. Moreover, using only one catalyst in several membranes is another interesting case to be investigated.

Acknowledgements

We gratefully acknowledge the most interesting discussions with Gheorghe Păun and other participants of the brainstorming week on P systems in Tarragona at the beginning of February 2003 as well as of the Fifth International Workshop Descriptional Complexitx of Formal Systems in Budapest, the stays of the Austrian authors being supported by MolCoNet project IST-2001-32008. The research of Petr Sosík partly was also supported by the Grant Agency of Czech Republic, grant No. 201/02/P079 and by the Canada Research Chair Program.

References

[1] Calude, C. S., Păun, Gh.: Computing with Cells and Atoms. Taylor & Francis, London (2001)

[2] Csuhaj-Varjú, E., Vaszil, Gy.: P Automata or Purely Communicating Accepting P Systems. In: [16] 219–233

[3] Dassow, J., Păun, Gh.: Regulated Rewriting in Formal Language Theory. Springer, Berlin (1989)

[4] Dassow, J., Păun, Gh.: On the Power of Membrane Computing. Journal of Universal Computer Science **5**, 2 (1999) 33–49

[5] Freund, R., Oswald, M.: GP Systems with Forbidding Context. Fundamenta Informaticae **49**, 1–3 (2002) 81–102

[6] Freund, R., Oswald, M., Sosík, P.: Reducing the Number of Catalysts Needed in Computationally Universal Systems without Priorities. In: Csuhaj-Varjú, E., Kintala, C., Wotschke, D., Vaszil, Gy. (Eds): Fifth International Workshop Descriptional Complexity of Formal Systems. Budapest, Hungary, July 12-14, 2003. MTA SZTAKI, Budapest (2003) 102–113

[7] Freund, R., Păun, Gh.: On the Number of Non-terminals in Graph-controlled, Programmed, and Matrix Grammars. In: Margenstern, M., Rogozhin, Y. (Eds.): Proc. Conf. Universal Machines and Computations, Chişinău (2001). Springer, Berlin (2001)

[8] Freund, R., Păun, Gh.: From Regulated Rewriting to Computing with Membranes: Collapsing Hierarchies. *To appear in TCS*

[9] Ibarra, O.: Characterizations of Catalytic Membrane Computing Systems. Proc. MFCS 2003 (2003)

[10] Krishna, S. N., Păun, A.: Three Universality Results on P Systems. In: Cavaliere, M., Martín-Vide, C., Păun, Gh. (Eds.): Brainstorming Week on Membrane Computing; Tarragona, Febr. 5-11, 2003. Rovira i Virgili University, Techn. Rep. No. 26 (2003) 198–206

[11] Minsky, M. L.: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, New Jersey (1967)

[12] Păun, Gh.: Computing with Membranes. Journal of Computer and System Sciences **61,** 1 (2000) 108–143

[13] Păun, Gh.: Computing with Membranes: an Introduction. Bulletin EATCS **67** (1999) 139–152

[14] Păun, Gh.: Membrane Computing: an Introduction. Springer, Berlin (2002)

[15] Păun, Gh., Rozenberg, G., Salomaa, A.: Membrane Computing with External Output. Fundamenta Informaticae **41**, 3 (2000) 259–266

[16] Păun, Gh., Rozenberg, G., Salomaa, A., Zandron, C. (Eds.): Membrane Computing. International Workshop, WMC-CdeA 2002, Curteă de Argeş, Romania, August 2002. LNCS 2597, Springer, Berlin (2003)

[17] Salomaa, A., Rozenberg, G. (Eds.): Handbook of Formal Languages. Springer, Berlin (1997)

[18] Sosík, P., Freund, R.: P Systems Without Priorities are Computationally Universal. In: [16] 400–409

[19] Sosík, P.: The Power of Catalysts and Priorities in Membrane Systems. Grammars **6**, 1 (2003) 13–24