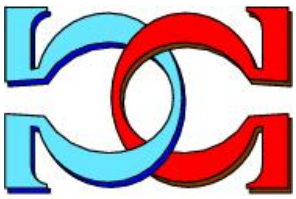


CDMTCS
Research
Report
Series

Deterministic Transition P Systems Modeled as Register Machines

Michael J. Dinneen
Yun-Bum Kim

Department of Computer Science,
University of Auckland,
Auckland, New Zealand

CDMTCS-465
July 2014

Centre for Discrete Mathematics and
Theoretical Computer Science

Deterministic Transition P Systems Modeled as Register Machines

Michael J. Dinneen and Yun-Bum Kim

Department of Computer Science, University of Auckland,
Private Bag 92019, Auckland, New Zealand

mjd@cs.auckland.ac.nz, tkim021@aucklanduni.ac.nz

July 28, 2014

Abstract

This paper presents the details for constructing register machines that simulate deterministic transition P systems with rule priorities. The time complexity of the constructed register machine is polynomial with respect to the number of rewriting rules applied. We illustrate our conversion with a non-trivial example.

Keywords: P systems, register machines, modeling, simulation.

1 Introduction

In theoretical computer science, there are many computational models that are equivalent to the computation power of a Turing machine. Two such models are register machines and, more recently, membrane systems. Register machines have been proposed in many flavors (mainly by theoreticians). These machines have a common theme of having a finite set of registers that can represent arbitrarily large non-negative integers. Also these machines are presented as a finite sequence taken from a small set of basic instructions (e.g. to do arithmetic, data handling and flow control). P systems [9, 11] (also called membrane systems) are distributed and parallel computing models, inspired by the structure and function of living cells. Several variants of P systems [8, 7] have been introduced, inspired from various features of living cells, that provide new ways to process information and solve computational problems of interest. Essentially, all P system models have a structure consisting of connected cells and a set of evolution rules that govern their evolution over time.

Several studies have investigated the relationship between P systems and register machines [6, 3] and presented universality results by proving that P systems can simulate a universal register machine [4]. Previously, in a companion paper [4], we presented the

details for constructing an efficient P system from an arbitrary register machine [1]. In this paper we address the opposite direction of mapping deterministic P systems to register machines. Our motivation is based on the fact that it is easier to design parallel algorithms using P systems instead of sequential-based classical (i.e. von Neumann architecture) computer models. Thus, we want to automate the conversion of a P system framework to the existing computers, which today have multi-core CPUs and many-core GPUs available. The work in this paper is a first step towards a translation to "parallel" register machines.

This paper is organized as follows. Section 2 recalls several key mathematical concepts that are used in this paper. Section 3 provides the definition of a transition P system model. Section 4 presents the definition of a register machine model. Section 5 presents the construction details for building a register machine that simulates a transition P system. Section 6 gives a non-trivial concrete mapping of a P system to a register machine. Finally, Section 7 summarizes this paper and provides some future areas of study.

2 Preliminaries

This section covers several key mathematical concepts that are used in this paper, such as sets, strings, multisets and graphs.

An *alphabet* is a finite non-empty set with elements called *symbols*. A *string* over alphabet O is a finite sequence of symbols from O. The set of all strings over O is denoted by O^*. The length of a string $x \in O^*$, denoted by $|x|$, is the number of symbols in x. The number of occurrences of a symbol $o \in O$ in a string x over O is denoted by $|x|_o$. The *empty string* is denoted by λ.

A multiset over an alphabet O is represented as strings over O, such as $o_1^{n_1} \dots o_k^{n_k}$, where $o_i \in O$ and $n_i \geq 0$, for $1 \leq i \leq k$. The *multiplicity* of an element x in a multiset v is denoted by $|v|_x$. We say that a multiset v is *included* in a multiset w, denoted by $w \subseteq v$, if, for all $o \in O$, $|w|_o \leq |v|_o$. The *union* of multisets v and w, denoted by $v \cup w$, is a multiset x, such that, for all $o \in O$, $|x|_o = |v|_o + |w|_o$. The *difference* of multisets v and w, denoted by $v - w$, is a multiset x, such that, for all $o \in O$, $|x|_o = \max(|v|_o - |w|_o, 0)$. The empty multiset is represented by λ. A set that contains the distinct elements of a multiset v is denoted by $\texttt{distinct}(v)$.

A (binary) *relation* R over two sets X and Y is a subset of their Cartesian product, $R \subseteq X \times Y$. For $A \subseteq X$ and $B \subseteq Y$, we set $R(A) = \{y \in Y \mid \exists x \in A, (x, y) \in R\}$, $R^{-1}(B) = \{x \in X \mid \exists y \in B, (x, y) \in R\}$.

A *directed graph* (digraph) is a pair (V, A), where V is a finite set of elements called nodes (or vertices), and A is a set of ordered pairs of V called *arcs*. Given a digraph $D = (V, A)$, for $v \in V$, the *parents* of v are $A^{-1}(v) = A^{-1}(\{v\})$ and the *children* of v are $A(v) = A(\{v\})$.

3 Transition P Systems

A deterministic transition P system (of order $n \geq 1$) is a construct of the form:

$$\Pi = (O, K, \Delta, W, R)$$

where:

1. O is the finite and non-empty alphabet of symbols.
2. $K = \{\sigma_i \mid 1 \leq i \leq n\}$ is the set of cells, where i represents the *cell ID* of σ_i.
3. Δ is an irreflexive and asymmetric relation on K, representing a set of arcs between cells with *bidirectional* communication capabilities, (i.e. parents can communicate to their children and vice versa).
4. $W = \{w_i \mid 1 \leq i \leq n\}$, where $w_i \in O^*$ is a multiset of symbols, called the *content*, present in cell σ_i.
5. $R = \{R_i \mid 1 \leq i \leq n\}$, where R_i is a finite set of evolution rules that are associated with cell σ_i. An evolution rule $r \in R_i$ is a *linearly ordered* transition multiset rewriting rule of the form:

   $$r: \; j \; u \rightarrow v$$

   where:

   - $j \in \{1, 2, \ldots, |R_i|\}$ indicates the *priority order* of r, where the lower value j indicates higher priority.
   - $u \in O^+$.
   - $v \in (O \times \tau)^*$, where $\tau \in \{\odot, \uparrow, \downarrow\}$ is a set of *target indicators*. Note that $(o, \odot) \in v$, $o \in O$, is abbreviated to o. Moreover, we denote:
     - multiset $v_\odot = \{o \mid (o, \odot) \in v\}$,
     - multiset $v_\uparrow = \{o \mid (o, \uparrow) \in v\}$ and
     - multiset $v_\downarrow = \{o \mid (o, \downarrow) \in v\}$.

Thanks to the unique priority assigned to each rule in the item 5 above, this transition P system is a deterministic model.

A cell *evolves* by applying one or more rules, which can change its content and can send multisets to its parent and child cells. For a cell $\sigma_i \in K$, a rule $j \; u \rightarrow v \in R_i$ is *applicable*, if $u \subseteq w_i$. The rules are applied in the *weak priority* order [10], i.e. higher priority applicable rules are applied, as many times as possible, before lower priority applicable rules. All applicable rules of all cells are applied simultaneously in one step. A computation *halts*, if none of the cells can evolve. The *output* of a halted transition P system computation is defined by the multiset of symbols present in the cells.

Applying an applicable rule $j \; u \rightarrow v$ in cell σ_i at step $k \geq 1$: (i) consumes multiset u at step k, i.e. $w_i = w_i - u$, (ii) produces multiset $v_\odot$, which will become available to σ_i at step $k + 1$, (iii) sends multiset $v_\uparrow$ to every parent cell $\sigma_p \in \Delta^{-1}(i)$ and sends multiset $v_\downarrow$

to every child cell $\sigma_c \in \Delta(i)$, which will become available to σ_p and σ_c at step $k+1$. We denote the multiset produced in cell σ_i in the current step by $\overline{w_i}$, i.e. $\overline{w_i} = \{v_\odot \mid j\ u \to v \in R_i\} + \{v_\uparrow \mid j\ u \to v \in R_c, \sigma_c \in \Delta(i)\} + \{v_\downarrow \mid j\ u \to v \in R_p, \sigma_p \in \Delta^{-1}(i)\}$. At the end of step k, σ_i updates its content as $w_i = w_i + \overline{w_i}$. The following pseudocode describes the behavior of the transition P system Π at each step $k \geq 1$. This pseudocode terminates when it reaches line 19.

```
1  bool evolve := false
2  for σ_i, i = 1, 2, ... |K|
3     for j = 1, 2, ..., |R_i|
4        r := (j u → v) ∈ R_i
5        while (u ⊆ w_i)
6           evolve := true
7           w_i := w_i − u
8           w̄_i := w̄_i + v_⊙
9           foreach σ_p ∈ Δ^{-1}(i)
10             w̄_p := w̄_p + v_↑
11          endfor
12          foreach σ_c ∈ Δ(i)
13             w̄_c := w̄_c + v_↓
14          endfor
15       endwhile
16    endfor
17 endfor
18 if (evolve = false) then
19    system Π halts
20 endif
20 foreach σ_i ∈ K
21    w_i := w_i + w̄_i
22    w̄_i := ∅
23 endfor
24 goto line 1
```

4 Register Machines

The register machine model used in this paper extends the register machine of [1] by adding an instruction that performs subtraction. A register machine has $n > 1$ instructions and $m > 0$ registers, where each register may hold an arbitrarily large non-negative integer.

A register machine program consists of a finite list of instructions, `EQ`, `SET`, `ADD`, `SUB`, `READ` and `HALT`, followed by an optional *input data*, denoted as a sequence of bits, with the restriction that the `HALT` instruction appears only once as the last instruction of the list. The first instruction of a program is indexed at address (i.e. line number) 0, and any value greater than or equal to n denotes the illegal branch error. In general, a register machine program is presented in: (i) *symbolic instruction* form or (ii) *machine instruction*

(i.e. raw binary) form. In this paper we adopt the symbolic instruction form, where labels of the form "L_x:" are added to make the presentation more readable.

A set of instructions of a register machine M, denoted in Chaitin's style [2], is described below. In the instructions below, variables z_1, z_2 and z_3 denote registers and k denotes a non-negative binary integer constant. The content of register z_i, $1 \leq i \leq 3$, is denoted by $\texttt{value}(z_i)$.

- Instruction $(\texttt{EQ},\ z_1,\ z_2,\ z_3)$ or $(\texttt{EQ},\ z_1,\ k,\ z_3)$:
  If $\texttt{value}(z_1) = \texttt{value}(z_2)$ or $\texttt{value}(z_1) = k$, then the execution of M continues at the $\texttt{value}(z_3)$-th instruction in the sequence. Otherwise, the execution of M continues at the next instruction.

- Instruction $(\texttt{SET},\ z_1,\ z_2)$ or $(\texttt{SET},\ z_1,\ k)$:
  $\texttt{value}(z_2)$ or the constant k is assigned to register z_1.

- Instruction $(\texttt{ADD},\ z_1,\ z_2)$ or $(\texttt{ADD},\ z_1,\ k)$:
  $\texttt{value}(z_1) + \texttt{value}(z_2)$ or $\texttt{value}(z_1) + k$ is assigned to register z_1.

- Instruction $(\texttt{SUB},\ z_1,\ z_2)$ or $(\texttt{SUB},\ z_1,\ k)$:
  $max\{\texttt{value}(z_1) - \texttt{value}(z_2), 0\}$ or $max\{\texttt{value}(z_1) - k, 0\}$ is assigned to register z_1.

- Instruction $(\texttt{READ},\ z_1)$:
  One bit is read into r_1, so the numerical value of z_1 becomes either 0 or 1. Any attempt to read past the last data-bit results in a run-time error.

- Instruction $(\texttt{HALT})$:
  This is the last instruction of the register machine program.

In the following, we will not use the `READ` instruction in our translation from P systems to register machines.

5 Translating P Systems into Register Machines

This section presents the details for building a register machine program I_{M_Π} for a register machine M_Π that simulates a deterministic transition P system $\Pi = (O, K, \Delta, W, R)$ of Section 3. The instructions of I_{M_Π} are in the symbolic instruction form, separated by white space.

We present two pseudocodes, side by side, with corresponding lines, where:

- **Left:** describes evolution of a transition P system Π (according to Section 3).

- **Right:** gives the details for building I_{M_Π}. The methods used in this pseudocode, such as `INITIALIZE`, `CONSUME`, `PRODUCE`, `APPLICABLE` and `EXECUTE`, are described in Sections 5.1, 5.2, 5.3, 5.4 and 5.5, respectively.

In a translated register machine, multisets are represented as follows. Register o_i stores the multiplicity of symbol $o \in O$ in cell $\sigma_i \in K$, i.e. multiset $\{o^{o_i} \mid o \in O\}$ equals w_i. Register $\overline{o_i}$ gives the multiplicity of symbol $o \in O$ to be stored into cell $\sigma_i \in K$ in the next step, i.e. multiset $\{o^{\overline{o_i}} \mid o \in O\}$ equals $\overline{w_i}$.

```
1
2  bool evolve := false
3  for σ_i, i = 1,2,... |K|
4      for j = 1,2,...,|R_i|
5          r := (j  u → v) ∈ R_i
6          while (u ⊆ w_i)
7              evolve := true
8              w_i := w_i − u
9              w̄_i := w̄_i + v_⊙
10             foreach σ_p ∈ Δ^{-1}(i)
11                 w̄_p := w̄_p + v_↑
12             endfor
13             foreach σ_c ∈ Δ(i)
14                 w̄_c := w̄_c + v_↓
15             endfor
16         endwhile
17     endfor
18 endfor
19 if (evolve = false) then
20     HALT
21 endif
22 foreach σ_i ∈ K
23     w_i := w_i + w̄_i
24     w̄_i := ∅
25 endfor
26 goto line 2
```

```
1  INITIALIZE()
2  append L_STEP: (SET, evolve, 0)
3  for σ_i, i = 1,2,...,|K|
4      for j = 1,2,...,|R_i|
5          r := (j  u → v) ∈ R_i
6          APPLICABLE(i, r, |R_i|)
7          append (SET, evolve, 1)
8          CONSUME(i, r)
9          PRODUCE(i, r, ⊙)
10         foreach σ_p ∈ Δ^{-1}(i)
11             PRODUCE(p, r, ↑)
12         endfor
13         foreach σ_c ∈ Δ(i)
14             PRODUCE(c, r, ↓)
15         endfor
16         append (EQ, a, a, L_R(i,j))
17     endfor
18 endfor
19
20 append L_R(|K|+1,1): (EQ, evolve, 0, L_HALT)
21
22 foreach σ_i ∈ K
23     EXECUTE(i)
24 endfor
25
26 append (EQ, a, a, L_STEP)
27 append L_HALT:(HALT)
```

5.1 INITIALIZE method

This method sets register o_i with the multiplicity of symbol $o \in O$ in cell $\sigma_i \in K$. For example, for cell σ_i with content $w_i = aabc \in O^*$, the values of registers a_i, b_i and c_i are 2, 1 and 1, respectively.

```
1  INITIALIZE()
2      foreach σ_i ∈ K
3          foreach o ∈ O
4              append (SET,  o_i,  |w_i|_o)
5          endfor
6      endfor
```

Proposition 1. `INITIALIZE` *appends* $|K| \cdot |O|$ *instructions.*

5.2 CONSUME method

This method implements $w_i := w_i - u$ of line 8, which corresponds to a cell consuming the multiset u.

```
1  CONSUME(cell_ID i, rule r = j u → v)
2      foreach o ∈ distinct(u)
3          append (SUB, o_i, |u|_o)
4      endfor
```

A difference of multisets operation $w_i := w_i - u$ transforms w_i, such that $|w_i|_o = |w_i|_o - |u|_o$ for each $o \in O$. An instruction (SUB, o_i, $|u|_o$), appended for each $o \in O$, subtracts the value $|u|_o$ to register o_i.

Proposition 2. *For a rule $r = j\ u \to v$,* CONSUME *appends* |distinct(u)| *instructions.*

5.3 PRODUCE method

This method implements $\overline{w_i} := \overline{w_i} + v_\tau$, $\tau \in \{\odot, \uparrow, \downarrow\}$, of lines 9, 11 and 14, which determines a multiset to be produced and stored in $\sigma_i \in K$.

```
1  PRODUCE(cell_ID i, rule r = j u → v, target τ)
2      foreach o ∈ distinct(v_τ)
3          append (ADD, o_i-bar, |v_τ|_o)
4      endfor
```

A union of multisets operation $\overline{w_i} := \overline{w_i} + v_\tau$ transforms $\overline{w_i}$, such that $|\overline{w_i}|_o = |\overline{w_i}|_o + |v_\tau|_o$ for each $o \in O$. An instruction (ADD, $\overline{o_i}$, $|v_\tau|_o$), appended for each $o \in O$, adds the value $|v_\tau|_o$ to register $\overline{o_i}$.

Proposition 3. *For a rule $r = u \to v$, with target indicator $\tau \in \{\odot, \uparrow, \downarrow\}$,* PRODUCE *appends* |distinct(v_τ)| *instructions.*

5.4 APPLICABLE method

This method, together with "append (EQ, $a, a, L_{\mathtt{R}(i,j)}$)" of line 16, implements the `while` statement of line 6, which involves a cell to check if its content contains the multiset specified on the left-hand side of a rule.

```
1  APPLICABLE(cell_ID i, rule r = j u → v, rulesetSize n)
2      append L_R(i,j):
3      foreach o ∈ distinct(u)
4          for m = 0, 1, ..., |u|_o − 1
5              if (j < n) then
6                  append (EQ, o_i, m, L_R(i,j+1))
7              else
8                  append (EQ, o_i, m, L_R(i+1,1))
```

```
10          endif
11        endfor
12    endfor
```

Condition $u \subseteq w_i$ is false, if there is a $o \in O$, such that $|u|_o > |w_i|_o$. For each $o \in \texttt{distinct}(u)$, APPLICABLE generates $|u|_o$ instructions below:

$$
\begin{aligned}
L_{\mathtt{R}(i,j)}\colon\ & (\texttt{EQ},\ o_i,\ 0,\ L)\\
& (\texttt{EQ},\ o_i,\ 1,\ L)\\
& (\texttt{EQ},\ o_i,\ 2,\ L)\\
& \vdots\\
& (\texttt{EQ},\ o_i,\ |u|_o - 1,\ L')
\end{aligned}
$$

which check the condition $\texttt{value}(o_i) \geq |u|_o$. If $\texttt{value}(o_i) \leq |u|_o - 1$, then, by one of these instructions, the execution continues to the line specified by the label L or L', which indicates the line number $k+1$, where line k contains instruction $(\texttt{EQ}, a, a, L_{\mathtt{R}(i,j)})$. If $\texttt{value}(o_i) \geq |u|_o - 1$ for all $o \in \texttt{distinct}(u)$, then the execution continues to the next instruction, and eventually, reaches instruction $(\texttt{EQ}, a, a, L_{\mathtt{R}(i,j)})$ that prompts an unconditional jump back to the line with the label $L_{\mathtt{R}(i,j)}$.

We note that a slight optimization in number of steps is possible if $|u|_o > 5$, where we can replace the sequence of $(\texttt{EQ},\ o_i, \dots)$ with a direct test of register machine instructions that check $\texttt{value}(o_i) < |u|_o$. However, in practice we believe rules have small $|u|_o$.

```
1   APPLICABLE(cell_ID i, rule r = j u → v, rulesetSize n)
2       append L_R(i,j): (SET, t1, |u|_o)
3       append (SUB, t1, 1)
4       append (SET, t2, |w_i|_o)
5       append (SUB, t2, t1)
6       if (j < n) then
7           append (EQ, t2, 0, L_R(i,j+1))
8       else
9           append (EQ, t2, 0, L_R(i+1,1))
10      endif
```

Proposition 4. *For a rule $r = j\ u \to v$,* APPLICABLE *will append at most* $\min(|u|, 5 \cdot |\texttt{distinct}(u)|)$ *instructions.*

5.5 EXECUTE method

This method implements $w_i := w_i + \overline{w_i}$ of line 23 and $\overline{w_i} := \emptyset$ of line 24, which represent cells updating their current content with the multiset produced from the execution of the rules.

```
1  EXECUTE(cell_ID i)
2      foreach o ∈ O
3          append (ADD, o_i, ō_i)
4          append (SET, ō_i, 0)
5      endfor
```

A union of multiset operation $w_i := w_i + \overline{w_i}$ transforms w_i, such that $|w_i|_o = |w_i|_o + |\overline{w_i}|_o$ for each $o \in O$. An instruction (ADD, o_i, $\overline{o_i}$), appended for each $o \in O$, adds the value of register $\overline{o_i}$ to register o_i. A multiset assignment operation $\overline{w_i} := \emptyset$ transforms w_i, such that $|\overline{w_i}|_o = 0$ for each $o \in O$. An instruction (SET, $\overline{o_i}$, 0), appended for each $o \in O$, sets the value of register $\overline{o_i}$ to 0.

Proposition 5. EXECUTE *appends* $2 \cdot |O|$ *instructions.*

6 Translation Example

We illustrate a non-trivial example for the following (deterministic) transition P system $\Pi_{\text{BFS}} = (O, K, \Delta, W, R)$. The system Π_{BFS}, starting with cell $\sigma_1 \in K$, visits all cells in breadth-first search (BFS) manner.

- $O = \{a, b\}$.
- $K = \{\sigma_1, \sigma_2, \ldots, \sigma_5\}$, where σ_1 represents the initiator.
- $\Delta = \{(\sigma_1, \sigma_2), (\sigma_1, \sigma_3), (\sigma_2, \sigma_4), (\sigma_2, \sigma_5), (\sigma_3, \sigma_2), (\sigma_3, \sigma_5), (\sigma_4, \sigma_1), (\sigma_4, \sigma_5), (\sigma_5, \sigma_4)\}$. Figure 1 (left) shows the membrane structure of the system Π_{BFS}.
- $w_1 = \{aab\}$ and $w_j = \{aa\}$, for $2 \leq j \leq 5$.
- R_i, $1 \leq i \leq 5$ is the set of evolution rules below.
  1. $a\ a\ b \to (b, \downarrow)$
  2. $b \to \lambda$

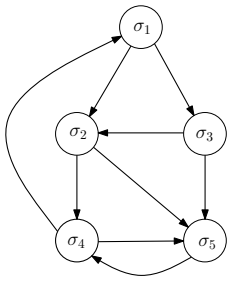
Initially, only σ_1 contains one copy of symbol b. By rule 1, when a cell contains symbol b, it sends one copy of symbol b to all its children. By rule 2, cells consume any additional copies of symbol b received from their parents. Note that these algorithmic rules work for alternative Δ structures.

Figure 1 (right) shows evolution trace, i.e. content of each cell at each step, of the system Π_{BFS}. Starting from cell σ_1, at each step $k \geq 0$, cells in level k with respect to σ_1 are visited (i.e. receive symbol b), e.g. cells σ_2 and σ_3 receive symbol b at step 1.

The following table contains the register machine program $I_{\Pi_{\text{BFS}}}$, which simulates the transition P system Π_{BFS}. $I_{\Pi_{\text{BFS}}}$ is generated by translating Π_{BFS} according to the pseudocode given in Section 5.

Line	Instruction
0	(SET, a_1, 2)
1	(SET, b_1, 1)
2	(SET, a_2, 2)
3	(SET, b_2, 0)
4	(SET, a_3, 2)
5	(SET, b_3, 0)
6	(SET, a_4, 2)
7	(SET, b_4, 0)
8	(SET, a_5, 2)
9	(SET, b_5, 0)
10	$L_{\texttt{STEP}}$: (SET, evolve, 0)
11	$L_{\texttt{R(1,1)}}$: (EQ, a_1, 0, $L_{\texttt{R(1,2)}}$)
12	(EQ, a_1, 1, $L_{\texttt{R(1,2)}}$)
13	(EQ, b_1, 0, $L_{\texttt{R(1,2)}}$)
14	(SET, evolve, 1)
15	(SUB, a_1, 2)
16	(SUB, b_1, 1)
17	(ADD, $\overline{b_2}$, 1)
18	(ADD, $\overline{b_3}$, 1)
19	(EQ, a, a, $L_{\texttt{R(1,1)}}$)
20	$L_{\texttt{R(1,2)}}$: (EQ, b_1, 0, $L_{\texttt{R(2,1)}}$)
21	(SET, evolve, 1)
22	(SUB, b_1, 1)
23	(EQ, a, a, $L_{\texttt{R(1,2)}}$)
24	$L_{\texttt{R(2,1)}}$: (EQ, a_2, 0, $L_{\texttt{R(2,2)}}$)
25	(EQ, a_2, 1, $L_{\texttt{R(2,2)}}$)
26	(EQ, b_2, 0, $L_{\texttt{R(2,2)}}$)
27	(SET, evolve, 1)
28	(SUB, a_2, 2)
29	(SUB, b_2, 1)
30	(ADD, $\overline{b_4}$, 1)
31	(ADD, $\overline{b_5}$, 1)
32	(EQ, a, a, $L_{\texttt{R(2,1)}}$)
33	$L_{\texttt{R(2,2)}}$: (EQ, b_2, 0, $L_{\texttt{R(3,1)}}$)
34	(SET, evolve, 1)
35	(SUB, b_2, 1)
36	(EQ, a, a, $L_{\texttt{R(2,2)}}$)
37	$L_{\texttt{R(3,1)}}$: (EQ, a_3, 0, $L_{\texttt{R(3,2)}}$)
38	(EQ, a_3, 1, $L_{\texttt{R(3,2)}}$)
39	(EQ, b_3, 0, $L_{\texttt{R(3,2)}}$)
40	(SET, evolve, 1)
41	(SUB, a_3, 2)
42	(SUB, b_3, 1)
43	(ADD, $\overline{b_2}$, 1)
44	(ADD, $\overline{b_5}$, 1)
45	(EQ, a, a, $L_{\texttt{R(3,1)}}$)
46	$L_{\texttt{R(3,2)}}$: (EQ, b_3, 0, $L_{\texttt{R(4,1)}}$)
47	(SET, evolve, 1)
48	(SUB, b_3, 1)
49	(EQ, a, a, $L_{\texttt{R(3,2)}}$)

Line	Instruction
50	$L_{\texttt{R(4,1)}}$: (EQ, a_4, 0, $L_{\texttt{R(4,2)}}$)
51	(EQ, a_4, 1, $L_{\texttt{R(4,2)}}$)
52	(EQ, b_4, 0, $L_{\texttt{R(4,2)}}$)
53	(SET, evolve, 1)
54	(SUB, a_4, 2)
55	(SUB, b_4, 1)
56	(ADD, $\overline{b_1}$, 1)
57	(ADD, $\overline{b_5}$, 1)
58	(EQ, a, a, $L_{\texttt{R(4,1)}}$)
59	$L_{\texttt{R(4,2)}}$: (EQ, b_4, 0, $L_{\texttt{R(5,1)}}$)
60	(SET, evolve, 1)
61	(SUB, b_4, 1)
62	(EQ, a, a, $L_{\texttt{R(4,2)}}$)
63	$L_{\texttt{R(5,1)}}$: (EQ, a_5, 0, $L_{\texttt{R(5,2)}}$)
64	(EQ, a_5, 1, $L_{\texttt{R(5,2)}}$)
65	(EQ, b_5, 0, $L_{\texttt{R(5,2)}}$)
66	(SET, evolve, 1)
67	(SUB, a_5, 2)
68	(SUB, b_5, 1)
69	(ADD, $\overline{b_4}$, 1)
70	(EQ, a, a, $L_{\texttt{R(5,1)}}$)
71	$L_{\texttt{R(5,2)}}$: (EQ, b_5, 0, $L_{\texttt{R(6,1)}}$)
72	(SET, evolve, 1)
73	(SUB, b_5, 1)
74	(EQ, a, a, $L_{\texttt{R(5,2)}}$)
75	$L_{\texttt{R(6,1)}}$: (EQ, evolve, 0, $L_{\texttt{HALT}}$)
76	(ADD, a_1, $\overline{a_1}$)
77	(SET, $\overline{a_1}$, 0)
78	(ADD, b_1, $\overline{b_1}$)
79	(SET, $\overline{b_1}$, 0)
80	(ADD, a_2, $\overline{a_2}$)
81	(SET, $\overline{a_2}$, 0)
82	(ADD, b_2, $\overline{b_2}$)
83	(SET, $\overline{b_2}$, 0)
84	(ADD, a_3, $\overline{a_3}$)
85	(SET, $\overline{a_3}$, 0)
86	(ADD, b_3, $\overline{b_3}$)
87	(SET, $\overline{b_3}$, 0)
88	(ADD, a_4, $\overline{a_4}$)
89	(SET, $\overline{a_4}$, 0)
90	(ADD, b_4, $\overline{b_4}$)
91	(SET, $\overline{b_4}$, 0)
92	(ADD, a_5, $\overline{a_5}$)
93	(SET, $\overline{a_5}$, 0)
94	(ADD, b_5, $\overline{b_5}$)
95	(SET, $\overline{b_5}$, 0)
96	(EQ, a, a, $L_{\texttt{STEP}}$)
97	$L_{\texttt{HALT}}$: (HALT)



Step	σ_1	σ_2	σ_3	σ_4	σ_5
0	a^2b	a^2	a^2	a^2	a^2
1		a^2b	a^2b	a^2	a^2
2		b		a^2b	a^2b^2
3	b			b	b
4					

Figure 1: Left: the membrane structure of the system Π_{BFS}. Right: evolution traces of the system Π_{BFS}.

7 Conclusions

The main result of this paper is a procedure that takes a transition P system and converts it to an equivalent register machine. Our approach can be described as follows. Assume that, at a particular step of a given P system, a membrane executes a maximal multiset of rules $Z = \{r_1^{k_1}, r_2^{k_2}, \ldots, r_j^{k_j} \mid k_i \geq 1, r_i \in R, 1 \leq i \leq j\}$. The converted register machine executes instructions that correspond to rule $r_i \in Z$, k_i times. Hence, the time complexity of the converted register machine is polynomial with respect to the number of rewriting rules applied. Our results open the following list of future work:

- Improve the time complexity of the converted register machine to polynomial with respect to the number of steps of the P systems by extending our register machine with *integer-division* and *multiplication* instructions.
- Develop a P system translator to a parallel register machine model that can efficiently be simulated by conventional parallel computers.
- Extend our preliminary results of [4] and the results of this paper by using more practical register machines and P systems, e.g. [8, 5]. P systems with active membranes [10] extends transition P systems by incorporating membrane handling rules that support *membrane creation* operation (which adds new cells to the system) and *membrane dissolution* operation (which removes existing cells from the system).

Acknowledgment

We would like to thank the three anonymous referees of our earlier version of this paper for providing useful comments that helped us to improve this paper. This paper was supported by the Quantum Computing Research Initiatives at Lockheed Martin.

References

[1] C. S. Calude and M. J. Dinneen. Exact approximations of Omega numbers. *Intl. J. of Bifurcation and Chaos*, 17(6):1937–1954, July 2007.

[2] G. J. Chaitin. *Algorithmic Information Theory.* Cambridge University Press, Cambridge, UK, 1987.

[3] E. Csuhaj-Varjú, M. Margenstern, G. Vaszil, and S. Verlan. On small universal antiport P systems. *Theor. Comput. Sci.*, 372(2-3):152–164, 2007.

[4] M. J. Dinneen and Y.-B. Kim. A new universality result on P systems. Report CDMTCS-423, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, Auckland, New Zealand, July 2012.

[5] M. J. Dinneen, Y.-B. Kim, and R. Nicolescu. A faster P solution for the Byzantine agreement problem. In M. Gheorghe, T. Hinze, and G. Păun, editors, *Conference on Membrane Computing*, pages 167–192. Verlag ProBusiness, Berlin, 2010.

[6] R. Freund, L. Kari, M. Oswald, and P. Sosík. Computationally universal P systems without priorities: two catalysts are sufficient. *Theor. Comput. Sci.*, 330(2):251–266, 2005.

[7] M. Ionescu, G. Păun, and T. Yokomori. Spiking neural P systems. *Fundam. Inform.*, 71(2-3):279–308, 2006.

[8] C. Martín-Vide, G. Păun, J. Pazos, and A. Rodríguez-Patón. Tissue P systems. *Theor. Comput. Sci.*, 296(2):295–326, 2003.

[9] G. Păun. *Membrane Computing: An Introduction.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[10] G. Păun. Introduction to membrane computing. In G. Ciobanu, M. J. Pérez-Jiménez, and G. Păun, editors, *Applications of Membrane Computing*, Natural Computing Series, pages 1–42. Springer-Verlag, 2006.

[11] G. Păun, G. Rozenberg, and A. Salomaa. *The Oxford Handbook of Membrane Computing.* Oxford University Press, Inc., New York, NY, USA, 2010.