

Encoding P System in Non-interleaving π -Calculus*

Cheng FU, Zhengwei QI, Jinyuan YOU

Dept. of Computer Science and Engineering
Shanghai Jiao Tong University, China
E-mail: {fucheng, qi-zw, you-jy}@cs.sjtu.edu.cn

Abstract

P systems and π -calculus are computational frameworks developed in different fields of computer science. In this paper, we establish a link between them by expressing P systems in non-interleaving π calculus. Our encoding solution shows that any P systems can be simulated in non-interleaving π calculus.

1 Introduction

P systems [9, 8] are computational models inspired from the structures and functioning of the biological membrane systems. Current intense study about P systems mainly covers computability, modelling, and encodings which leads to many variants. One can find the newest results and open problems as well as guides to P systems in its web site (<http://psystems.disco.unimib.it>). To make them more realistic, some efforts have been made to simulate P systems in formalized P systems[2, 6], mobile ambients [7], and Petri Nets [11]. In this paper, we use π calculus to formally describe P systems.

Π calculus [5, 12] was first introduced in 1992 to serve as a fundamental framework for parallel and distributed system. It plays important role in programming language and application modelling in concurrent and communication systems. Due to the rapid advancements of the Internet technology, the original calculus finds some obstacles in the area of security, mobility, and structured systems. Many variants [3, 4, 1] from core π calculus have then been studied in order to enhance these features. Along these variants, only non-interleaving semantics for π calculus have the structured expressive power. This naturally brings with the idea to express membrane systems in non-interleaving π calculus.

The rest of the paper is organized as follows: Section 2 and 3 present reviews of P systems and π calculus. Section 4 presents the formalization of P systems in π calculus with a illustrative example. And finally we have the conclusion in section 5.

*Project is supported by the Shanghai Science and Technology Development Foundation (No. 03DZ15027) and the National Natural Science Foundation of China (No. 60173033).

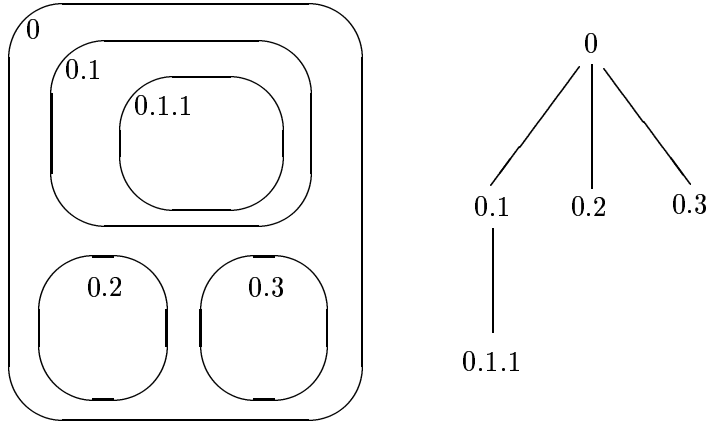


Figure 1: Membrane Identity Renumbering

2 P Systems

Here we give a brief review of P systems. A full guide for P systems can be referred to [10]. In general, a basic P system Π (of degree $m \geq 0$) is composed of alphabets of objects (V), outputs (T) and catalysts (C), a membrane structure (μ), and the sets of corresponding membrane objects (ω_i), rules (R_i), and priorities (ρ_i) where $i = 0, 1, \dots, m-1$. In membrane structure μ , membranes are identified by an integer i . From such identity we may not figure out the hierarchy of the system of membranes except that 0 always denote the root membrane, that's why μ uses a intuitional representation by hierarchical structures. To technically simplify the hierarchical structure encoding, we use a new form of identity according to their nested arrangement, for instance, $[0[1[2]2]1[3]3[4]4]_0$ is relabelled as $[0[0.1[0.1.1]0.1.1]0.1[0.2]0.2]0.2[0.3]0.3]_0$ (Fig. 1), or more tersely, $\mu = \{0.1.1, 0.3\}$. Each period separates different membrane at different hierarchical location just like outline numbered style in textbooks. We denote by θ such kind of identity, and its syntax is defined as:

$$\theta ::= 0 \mid \theta.d, \quad d \in \mathbb{Z}^+$$

For example, if $\theta = \theta'.i$ then membrane θ is a child of membrane θ' . The numbering scheme of membranes, from the perspective of a tree, proceeds top to bottom and left to right. If a parent membrane is numbered as θ , then for each immediate child of θ , their identities are sequentially numbered by $\theta.1, \theta.2$, and so forth. And, in a more tersely manner, a membrane structure μ can be denoted by a set of the membrane identities with the largest number at each level. For instance, $\{0.5\}$ indicates a root membrane 0 with five immediate children (0.1, 0.2, 0.3, 0.4, 0.5) in it because the 0.5 cannot exist without the existence of 0.4; $\{0.1.3, 0.3.4\}$ shows four children membranes inside 0.3, three children inside 0.1, and three children (0.1, 0.2, 0.3) inside the root. A function $inf(\mu)$ is defined to return a set all identities that can be inferred from μ . Hereafter we call μ the set of membrane identities.

After we adjust the identifying mechanism of membranes, now we have a rectified

formal definition of P system.

Definition 2.1 A P system (of degree $m \geq 0$) is a tuple

$$\Pi = (V, C, T, \mu, \Sigma)$$

where

1. V is an alphabet of objects;
2. $T(\subseteq V)$ is the output alphabet;
3. $C(\subseteq V)$ is the catalyst alphabet where $C \cap T = \emptyset$;
4. μ is a set of membrane identities where $\text{inf}(\mu) = m$;
5. Σ is a set of membrane contents, each element σ_θ of Σ is a triplet

$$\sigma_\theta = (\omega_\theta, R_\theta, \rho_\theta) \quad \theta \in \text{inf}(\mu)$$

where

- (a) ω_θ is a string over V associated with the regions of membrane θ , they represent multisets of objects present in the regions of θ ;
- (b) R_θ is a set of evolution rules associated with membrane θ . An *evolution rule* is pair (u, v) of strings over V , and the objects from v also have associated target indications of the form *here*, in_θ , and *out*;
- (c) ρ_θ is a partial order over R_θ , called a priority relation.

3 π Calculus

In this section, we briefly introduce the non-interleaving semantics for π calculus. For basic concepts related to core π calculus, [5] is recommended. There are a few kinds of non-interleaving semantics for π calculus. Anyway, they basically have one thing in common: the commutative law for parallel operator is dropped. Therefore, the syntactical tree of parallel operator can be fixed. This provides a way to encode structured systems.

Definition 3.1 Let \mathcal{N} be a countable infinite set of *names* ranged over by a, b, \dots and x, y, \dots . *Processes* (denoted by $P, Q, R, \dots \in \mathcal{P}$) are built from names according to the syntax

$$P, Q ::= \mathbf{0} \mid \pi.P \mid P + Q \mid P \mid Q \mid (vx)P \mid [x = y]P$$

where π may be either $x(y)_{@a}$ for *input*, or $\bar{x}(y)_{@a}$ for *output*, or τ for *silent* moves. $@a$ is called *action firing address*. And the trailing $\mathbf{0}$ will be omitted.

We recall the notion of the free names $fn(\pi)$, bound names of $bn(\pi)$ and names $n(\pi)$ of π . And functions fn, bn and n are extended to processes in the obvious way. The *structural congruence* \equiv on processes is defined as the least congruence satisfying:

- $P \equiv_\alpha Q \Rightarrow P \equiv Q$;
- $(\mathcal{P}/\equiv, +, \mathbf{0})$ is a commutative monoid;
- $[x = x]P \equiv P$;
- $(vx)(vy)P \equiv (vy)(vx)P$, $x \notin fn(Q) \Rightarrow (vx)(P|Q) \equiv (vx)P|Q$,
 $x \notin fn(P) \Rightarrow (vx)(P|Q) \equiv P|(vx)Q$, and $x \notin fn(P) \Rightarrow (vx)P \equiv P$,
 $\mathbf{0}|P \equiv P$, $\mathbf{0}|\mathbf{0} \equiv \mathbf{0}$.

Definition 3.2 Let $\vartheta \in \{\|_0, \|_1\}^*$. Then *proof terms* (denoted by ψ) are defined by the following syntax:

$$\psi ::= \vartheta\pi \mid \vartheta\langle \|_0\vartheta_0\pi_0, \|_1\vartheta_1\pi_1 \rangle$$

with $\pi_0 = x(z)_{\textcircled{\vartheta_0}} \Leftrightarrow \pi_1$ is $\bar{x}(y)_{\textcircled{\vartheta_1}}$, or vice versa.

Function la is defined as $la(\vartheta\pi) = \pi$ and $la(\vartheta\langle \|_0\vartheta_0\pi_0, \|_1\vartheta_1\pi_1 \rangle) = \tau$. Then we show the proved transition system below:

$$\begin{aligned} \text{Act:} \quad & \pi.P \xrightarrow{\pi} P \quad \text{Sum:} \quad P \xrightarrow{\psi} P' \Rightarrow P + Q \xrightarrow{\psi} P' \\ \text{Par0:} \quad & P \xrightarrow{\psi} P' \wedge bn(la(\psi)) \cap fn(Q) = \emptyset \Rightarrow P|Q \xrightarrow{\|_0\psi} P'|Q \\ \text{Par1:} \quad & P \xrightarrow{\psi} P' \wedge bn(la(\psi)) \cap fn(Q) = \emptyset \Rightarrow Q|P \xrightarrow{\|_1\psi} Q|P' \\ \text{Res:} \quad & P \xrightarrow{\psi} P' \wedge x \notin n(la(\psi)) \Rightarrow v(x)P \xrightarrow{\psi} (vx)P' \\ \text{Com0:} \quad & P \xrightarrow{\vartheta_1\bar{x}(y)_{\textcircled{\vartheta_0}}} P' \wedge Q \xrightarrow{\vartheta_2x(z)_{\textcircled{\vartheta_0}}} Q' \Rightarrow P|Q \xrightarrow{\langle \|_0\vartheta_1\bar{x}(y), \|_1\vartheta_2x(z) \rangle} P'|Q'\{y/z\} \\ \text{Com1:} \quad & P \xrightarrow{\vartheta_1x(z)_{\textcircled{\vartheta_0}}} P' \wedge Q \xrightarrow{\vartheta_2\bar{x}(y)_{\textcircled{\vartheta_0}}} Q' \Rightarrow P|Q \xrightarrow{\langle \|_0\vartheta_1x(z), \|_1\vartheta_2\bar{x}(y) \rangle} P'\{y/z\}|Q' \end{aligned}$$

By proved semantics, each action can be traced at a certain location in the syntactic tree where each node represents a sub process in that tree. For further concern about the proved semantics for bounded output, one can refer to [3, 4]. Now we move to the next section to encoding P systems.

4 Encoding P Systems

4.1 Encoding Membrane Structures

At this moment, for P systems, all objects and rules of each membrane are abstracted as a single *membrane process*, and their encoding will be introduced in the next section. As a example of Fig. 1, we assume each membrane process P_θ are respectively located in membrane θ where $\theta = 0, 0.1, 0.1.1, 0.2$, and 0.3 . Then this nested structure (see Fig. 2(d) by compared with membrane structure $\{0.1\}, \{0.2\}, \{0.3\}$), is encoded into a binary parallel operator syntax tree in the causal π calculus where P_0 is put at the address $\|_0$, $P_{0.1}$ at $\|_1\|_1\|_0$, $P_{0.1.1}$ at $\|_1\|_1\|_1\|_1$, $P_{0.2}$ at $\|_1\|_0\|_1\|_1$, and $P_{0.3}$ at $\|_1\|_0\|_1\|_0\|_1\|_1$. And the parent node of each membrane process is called *membrane node*. We define function $pa : \vartheta \rightarrow \vartheta'$ such that $\vartheta = \vartheta'\|_0$ or $\vartheta = \vartheta'\|_1$,

that is to compute the address of the parent node of ϑ . Let $loc_{P_\theta} \in \{\|_0, \|_1\}^*$ denote the location (address) of the subscript process in the syntactical tree.

More generally, we have two ways of the encoding solution for membrane structure.

Definition 4.1 Let P_θ be a membrane process in membrane θ , then

1. If $\theta = 0$, then $loc_{P_\theta} = \|_0$;
2. If $\theta = \theta'.i$, and for some i , then $loc_{P_\theta} = pa(loc_{P_{\theta'}})(\|_1)^i \|_0 \|_0$;
3. Other leaves are marked by nil process (**0**).

The definition shows a way to encoding membrane numbers (identities) to the absolute address (leaves) of a parallel operator syntax tree. Let $loc_\theta \triangleq loc_{P_\theta}$ to simply represent the address encoding of the membrane number θ . Each membrane process P_θ is put to the leaf at address loc_θ .

The second solution uses a different recursive definition. To simplify the presentation, we first define a nested context: $\mathcal{C}_{1,P_1}(-) \triangleq P_1|(-)$ and $\mathcal{C}_{k,P_k}(-) \triangleq \mathcal{C}_{k-1,P_{k-1}}(P_k|(-))$ where $k \geq 2$.

Definition 4.2 Let P_θ be a membrane process in membrane θ of a P system Π , then P_Π is said to be a *process with respect to Π* where

1. $P_\Pi = P_0|P_0^m$;
2. $P_\theta^m = \mathcal{C}_{k,P_{\theta,k}^a}(\mathbf{0})$ where there are totally k sub-membranes in θ ;
3. $P_\theta^a = P_\theta|P_\theta^m$;

This approach shows an algebra way to directly construct a process from a P system. Intuitively, process P_θ^m represents all sub-membranes inside θ , and process P_θ^a contains a membrane process P_θ as well as P_θ^m . The context $\mathcal{C}_{k,P_k}(-)$ just builds a membrane process structurally according to Def 4.1. The following result shows that both of the above solutions (Def. 4.1 and 4.2) coincide with each other.

Proposition 4.1 Let P_θ be a membrane process in membrane θ of a P system Π , then the parallel operator syntax tree of process P_Π coincides with the syntax tree constructed by Def. 4.1.

4.2 Encoding Objects and Rules

So far, we denote all except nested membranes by a single process in a membrane. Now we begin to encode objects and rules. For objects, each of them is encoded as a solo process which only outputs an arbitrary name through a channel representing that object, e.g. object a is encoded as $O_a = \bar{a}_{loc_\theta}(x)$ if the current object is inside membrane θ . Then all encoded processes of all objects in membrane θ combines a whole process $O_\theta = O_{obj_1}|O_{obj_2}|\dots|O_{obj_n}$.

Object encoding is very simple, but for rules, it is a little more complex as they have *in*, *here*, *out*, and δ communications, and even to be cooperative and

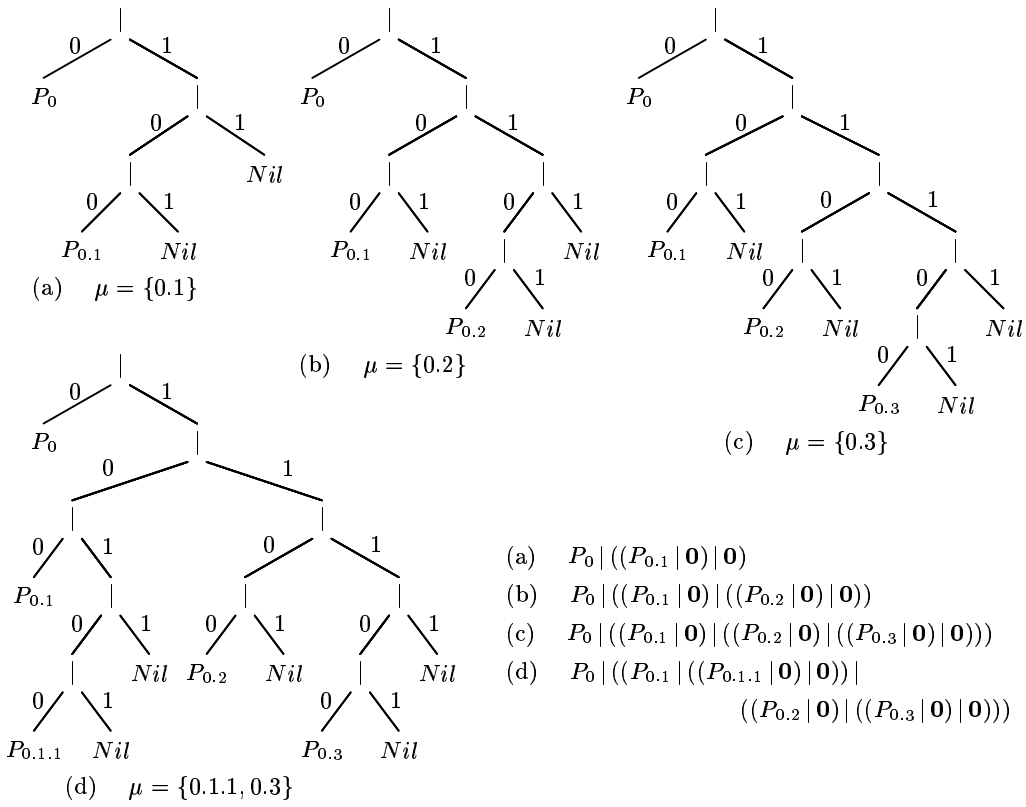


Figure 2: Encoding Membrane Structure

privileged. We leave the discussion of dissolution operation, and cooperative and privileged rules for the later sections, here for non-cooperative rules, a helper process $H_\theta \triangleq ch(y)_{@ps(cur)} \cdot (\bar{y}\langle x \rangle_{@ps(cur)} | H_\theta)$ is added for each membrane to receive the objects from parents and children membranes after encoding, where θ is the number of current membrane where function cur returns the address of the current process and function ps returns the address of the parent membrane node of the parameter.

All encoded processes of all rules in membrane θ combine a whole process $R_\theta = R_\theta^1 + R_\theta^2 + \dots + R_\theta^n$. The encoded process of each rule in membrane θ is recursively defined by R_θ , e.g. rule $a \rightarrow a_{here}b_{here}$ is encoded as

$$R_\theta^1 = a(x)_{@ps(cur)} \cdot ((\bar{a}\langle x \rangle_{@ps(cur)} | \bar{b}\langle x \rangle_{@ps(cur)}) | R_\theta)$$

If a rule tends to put some object into its child membrane, e.g. $a \rightarrow a_{in \theta, i}b_{here}$, then this rule is encoded as

$$R_\theta^2 = a(x)_{@ps(cur)} \cdot ((\bar{c}_{\theta, i}\langle a \rangle_{@ps(cur)} | \bar{b}\langle x \rangle_{@ps(cur)}) | R_\theta)$$

In this process, $\bar{c}_{loc\theta, i}\langle a \rangle$ will send name a to the helper process $H_{\theta, i}$ and produce O_a in the subprocess $P_{\theta, i}$. For rules containing out operation, e.g. $a \rightarrow a_{out}b_{here}$, we have the encoding as

$$R_\theta^3 = a(x)_{@ps(cur)} \cdot ((\bar{c}\langle a \rangle_{@ps(ps(cur))} | \bar{b}\langle x \rangle_{@ps(cur)}) | R_\theta)$$

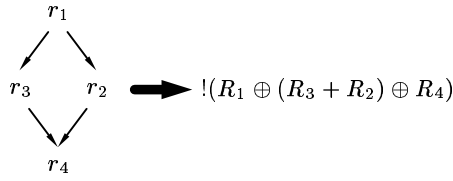


Figure 3: Encoding Prioritized Rules

So, a subprocess P_θ is composed of sequential composite processes of its encoded objects and rules of membrane θ .

4.3 Encoding Prioritized Rules

In P systems rules can have prioritized relation between them such that low-priority rule can be applied if and only if the higher one meets the condition to be applied. Here, we equip the π calculus with *prioritized choice operator* (\oplus). If we have rules $r_1 > r_2 > \dots > r_k$ in membrane θ , then the result of encoding becomes $R_1 \oplus R_2 \oplus \dots \oplus R_k$ which shows that a process R_i will be executed, where i ($1 \leq i \leq k$) is the minimal such that $R_i \downarrow_a$ and there exists O_a in subprocess P_θ .

Since the prioritized relation $>$ is a partial order over the set (\mathbb{R}_θ) of all rules of membrane θ , we have to solve the situation that rules priorities are interleaving. Consider rules $r_1 > r_2 > r_4$ and $r_1 > r_3 > r_4$, which forms a diamond, and the appropriate encoding is shown in Fig. 3. Below we define an encoding solution for prioritized choice \oplus .

Definition 4.3 Let r_θ^i be the rules in membrane θ indexed by i , and R_θ^i be the corresponding translated process in π calculus, moreover we use PR_θ to denote the combination ($\oplus/+$) process of multiple rule processes of membrane θ , then

1. $r_\theta^i > r_\theta^j \implies R_\theta^i \oplus R_\theta^j$
2. $PR_{\theta,1} \oplus PR_{\theta,2} \wedge PR_{\theta,2} \oplus PR_{\theta,3} \implies PR_{\theta,1} \oplus PR_{\theta,2} \oplus PR_{\theta,3}$
3. $PR_{\theta,1} \oplus PR_{\theta,3} \wedge PR_{\theta,2} \oplus PR_{\theta,3} \implies (PR_{\theta,1} + PR_{\theta,2}) \oplus PR_{\theta,3}$
4. $PR_{\theta,1} \oplus PR_{\theta,2} \wedge PR_{\theta,1} \oplus PR_{\theta,3} \implies PR_{\theta,1} \oplus (PR_{\theta,2} + PR_{\theta,3})$

During translation, all rules will be encoded to the corresponding rule processes at first. After that, we use the 1st regulation in Def. 4.3 to translate raw priorities to raw prioritized processes in π calculus. Then we use the next three regulations to reduce raw prioritized processes to combination processes for those rules until, for each $R_{\theta,i}$ in PR_θ , all rule processes where the corresponding rules have higher priorities than $r_{\theta,i}$, appear left to the process $R_{\theta,i}$. Finally, all remaining processes combine a whole process $R_\theta = PR_{\theta,1} + PR_{\theta,2} + \dots + PR_{\theta,n}$.

As an example, the translated raw prioritized processes of Fig. 3 are $R_1 \oplus R_3, R_1 \oplus R_2, R_3 \oplus R_4, R_2 \oplus R_4$. Then we derive $R_1 \oplus (R_3 + R_2)$ and $(R_3 + R_2) \oplus R_4$ by 3rd and 4th regulations of Def. 4.3. Then by 2nd regulation, we have $(R_1 \oplus (R_3 + R_2)) \oplus R_4$.

Another possible translation is $(R_1 \oplus R_2 \oplus R_4) + (R_1 \oplus R_3 \oplus R_4)$ because of different derivation sequences. Both of the results coincide with the original rules priorities.

For more complicated rule relationships, the translation should be more carefully derived. For example, if we add $r_5 > r_3$ to Fig. 3. Then the only possible result should be $((R_1 + R_5) \oplus R_3 \oplus R_4) + (R_1 \oplus R_2 \oplus R_4)$. But if we derive in other direction, the non-continuable result will be $R_5 > R_3$ and $R_1 \oplus (R_3 + R_2) \oplus R_4$. A better way which leading to a legal result is to derive from bottom up to top in a rule graph.

4.4 Encoding Membrane Dissolution

In P system, the dissolution of membrane $\theta.i$ causes all objects and sub membranes to enter membrane θ , and all rules in $\theta.i$ are removed. In π calculus, this operation can be expressed by cutting sub-processes (sub-trees) of a parallel operator syntax tree and link them to other nodes. The following lemmas prepares for the encoding of dissolve rules.

Lemma 4.1 *For an encoded process P_Π , any leaf of its parallel syntax tree is an offspring of at least one membrane node.*

Proof. Assume a leaf process P_l and its location ϑ_l . By Def 4.1, we can show that P_l belongs to at least one membrane node θ , that is, for some θ we have ϑ such that $loc_{P_l} = pa(P_\theta)\vartheta$, denoted by $P_l \triangleleft pa(P_\theta)$. ■

Lemma 4.2 *Let P_l be a leaf of the parallel syntax tree of the encoded process P_Π , if $P_l \triangleleft pa(P_\theta)$ and $P_l \triangleleft pa(P_{\theta'})$, then there exists a ϑ such that $pa(P_\theta) = pa(P_{\theta'})\vartheta$ or $pa(P_{\theta'})\vartheta = pa(P_\theta)$*

Proof. We can show that there is a path from top to bottom leaf P_l such that all membrane nodes of which P_l is an offspring, are located in that path. ■

A *parent membrane node* of P_l is a membrane node θ such that $P_l \triangleleft pa(P_\theta)$ and for any other membrane node θ' , $P_l \triangleleft pa(P_{\theta'}) \implies pa(P_\theta) = pa(P_{\theta'})\vartheta$ for some ϑ . The address of a parent membrane node of P located at address ϑ is denoted by $pm(\vartheta)$.

A dissolve rule, e.g. $a \rightarrow b_{here}\delta$, is encoded as

$$R = a(x)_{@ps(cur)}.(b(x) | Dis)$$

where process Dis cuts all children of its parent membrane node and link them to other nodes: all rule processes in P_θ are dropped; all object processes are moved to be parallel to its parent membrane process $P_{\theta'}$; all child membrane nodes of current membrane node are cut to append other sub processes in the parent membrane node. For example, the syntax tree of the encoded process of Fig. 1 is shown in Fig. 4(a) where P_θ^m represents the process containing all sub membranes in θ . If a dissolve rule is applied in $P_{0.1}$, then the whole process will evolve as (Fig. 4(b))

$$(P_0 | P'_{0.1}) | ((P_{0.2} | P_{0.2}^m) | ((P_{0.3} | P_{0.3}^m) | P_{0.1}^m))$$

In the result, membrane process $P'_{0.1}$ moves to the location at $\|_0\|_1$, and $P_{0.1}^m$ moves to the location at $\|_1\|_0\|_1$. Furthermore, to coincide with the concept of dissolution

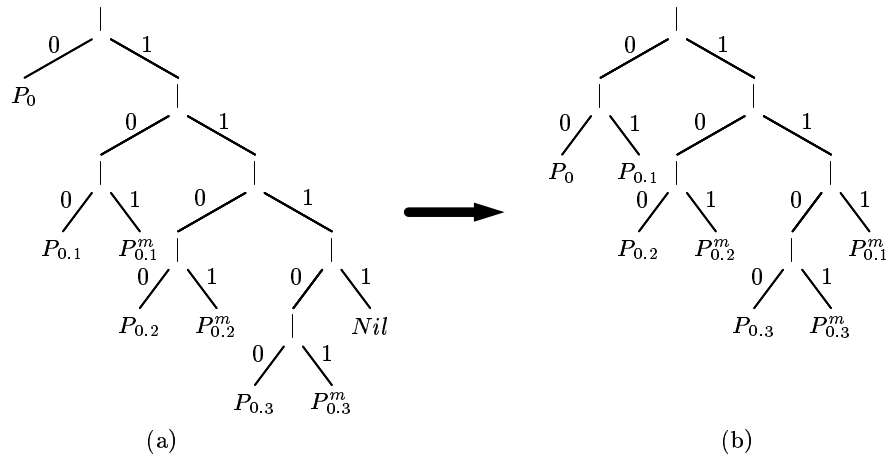


Figure 4: Dissolve in $P_{0,1}$

in P system, for process $P'_{0,1}$, all rule processes are consumed, and only object processes remain.

Process Dis comprises two sub-processes. One (Dis_1) takes the task to remove all rule processes in the current membrane, this is implemented by a summation of rule combination processes and an input process, the latter one only accepts an input data, and then all rules are removed:

$$PR' \triangleq PR + dis(x)$$

$$Dis_1 \triangleq \overline{dis}(y)$$

The other sub-process (Dis_2) takes the task of cutting and relinking in the parallel operator syntax tree. We introduce a move action $\vartheta_1 \mapsto \vartheta_2$ to handle such activity. This action causes the sub-process, e.g. P_m , at location ϑ_1 to be repositioned at ϑ_2 . Before moving, there must be a process, e.g. P_t , (or may be a nil process) at the target location ϑ_2 . And after moving, e.g. the subprocess at the target location becomes $P_t | P_m$; and a nil process ($\mathbf{0}$) is put at the source location ϑ_1 . Moreover, move action must be exercised inside a process tending to move. For example:

$$(P_1 \parallel \mathbf{0} \parallel_1 \mapsto \parallel_1.P_2) | P_3 \longrightarrow (P_1 | \mathbf{0}) | (P_3 | P_2)$$

$$(P_1 \parallel \mathbf{0} \mapsto \parallel_1.P_2) | P_3 \longrightarrow \mathbf{0} | (P_3 | (P_1 | P_2))$$

A reduction rule is added here for parallel move action:

$$\frac{P \xrightarrow{\vartheta_1 \mapsto \vartheta_2} P' \quad P@_{\vartheta_1} \circ \mathcal{C}(P, Q) \quad Q@_{\vartheta_2} \circ \mathcal{C}(P, Q)}{\mathcal{C}(P, Q) \longrightarrow \mathcal{C}(\mathbf{0}, Q | P')}$$

In the above rule, $P@_{\vartheta_1} \circ \mathcal{C}(P, Q)$ represents sub-process P is located at the address ϑ_1 of the process $\mathcal{C}(P, Q)$. Context $\mathcal{C}(x_1, x_2)$ is parameterized by two process variables, and $\mathcal{C}(P, Q) \equiv \mathcal{C}(x_1, x_2)\{P/x_1, Q/x_2\}$. As an example in Fig. 4, process Dis_2 , is expressed as

$$ps(cur) \parallel_1 \mapsto \parallel_1 \parallel_1 \parallel_1 \parallel_1.ps(cur) \parallel_0 \mapsto ps(ps(cur)) \parallel_0$$

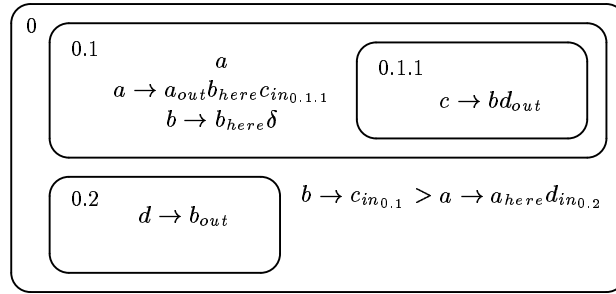
The address referenced in the above expression is called *absolute address reference* where each address is calculated from the root node. And the result of $ps(cur)$ is $\|_1\|_0$, $ps(ps(cur))$ is ϵ . For the encoded P system process in π calculus, dissolve activity basically has a fixed move direction in a relative manner. As an example in Fig. 4, the encoded process $P_{0.1}^m$ moves from the address $\|_1\|_0\|_1$ to $\|_1\|_1\|_1\|_1$ while, in relative manner, we pickup the same address prefix, and the action can be simply written as $pa(ps(cur))(\|_0\|_1 \mapsto \|_1\|_1\|_1)$ where the result of $pa(ps(cur))$ is $\|_1$. Moreover, since after dissolution of θ , all unleashed sub-membranes (encoded as P_θ^m), are always positioned at rightmost and bottommost leaf node (has an address form of $(\|_1)^*$) under the grandfather $(\|_0\|_1)$ of the current process P_θ^m , then a more simplified form is used: $pa(ps(cur))(\|_0\|_1 \mapsto (\|_1)^e)$ where $(\|_1)^e$ means the possibly longest address path from the current node in the tree. Now, in a more elegant manner, the process Dis_2 is generally modelled as

$$pa(ps(cur))(\|_0\|_1 \mapsto (\|_1)^e).ps(cur)\|_0 \mapsto ps(ps(cur))\|_0$$

Thus we can use $Dis \triangleq Dis_1 | Dis_2$ to serve as a general process for dissolution operation during the encoding.

4.5 Example

Here presents a simple but illustrative example of encoding. Consider the following P system of degree 4 (identity renumbered):



$$\begin{aligned} \Pi &= (\{a, b, c, d\}, \emptyset, \emptyset, \mu, \{\sigma_0, \sigma_{0.1}, \sigma_{0.1.1}, \sigma_{0.2}\}), \\ \mu &= \{0.1.1, 0.2\}, \quad (\text{abbr. of } \{0, 0.1, 0.1.1, 0.2\}) \\ \sigma_0 &= (\emptyset, \{r_0^1 : b \rightarrow c_{in_{0.1}}, r_0^2 : a \rightarrow a_{here} d_{in_{0.2}}\}, \{r_0^1 > r_0^2\}), \\ \sigma_{0.1} &= (\{a\}, \{a \rightarrow a_{out} b_{here} c_{in_{0.1.1}}, b \rightarrow b_{here} \delta\}, \emptyset), \\ \sigma_{0.1.1} &= (\emptyset, \{c \rightarrow b d_{out}\}, \emptyset), \\ \sigma_{0.2} &= (\emptyset, \{d \rightarrow b_{out}\}, \emptyset) \end{aligned}$$

The encoding solution for the initial configuration of Π is shown below:

$$\begin{aligned} P_\Pi &= P_0 | ((P_{0.1} | ((P_{0.1.1} | \mathbf{0}) | \mathbf{0})) | ((P_{0.2} | \mathbf{0}) | \mathbf{0})) \\ P_0 &= H_0 | R_0 \quad R_0 = R_0^1 \oplus R_0^2 \end{aligned}$$

$$\begin{aligned}
R_0^1 &= b(x)_{@ps(cur)} \cdot (\overline{ch}_{0.1} \langle c \rangle_{@ps(cur)} | R_0) \\
R_0^2 &= a(x)_{@ps(cur)} \cdot (O_a | (\overline{ch}_{0.2} \langle D \rangle_{@ps(cur)} | R_0)) \\
P_{0.1} &= H_{0.1} | O_{0.1} | R_{0.1} \quad O_{0.1} = O_a \quad R_{0.1} = R_{0.1}^1 + R_{0.1}^2 + Dis(x) \\
R_{0.1}^1 &= a(x)_{@ps(cur)} \cdot (\overline{ch} \langle a \rangle_{@ps(ps(cur))} | (O_b | (\overline{ch}_{0.1.1} \langle c \rangle_{@ps(cur)} | R_{0.1}))) \\
R_{0.1}^2 &= b(x)_{@ps(cur)} \cdot (O_b | R_{0.1}) \\
P_{0.1.1} &= H_{0.1.1} | R_{0.1.1} \\
R_{0.1.1} &= c(x)_{@ps(cur)} \cdot (\overline{ch} \langle b \rangle_{@ps(ps(cur))} | (\overline{ch} \langle d \rangle_{@ps(ps(cur))} | R_{0.1.1})) \\
P_{0.2} &= H_{0.2} | R_{0.2} \quad R_{0.2} = d(x)_{@ps(cur)} \cdot (\overline{ch} \langle b \rangle_{@ps(ps(cur))} | R_{0.2})
\end{aligned}$$

The solution process P_Π preserves the membrane structure by means of Def. 4.1 and 4.2. Membrane process P_θ is the encoding result of σ_θ , and R_θ is the encoding result of all rules in membrane θ . A dissolution channel appears in process $R_{0.1}$ for there is a dissolve rule in membrane 0.1. And $O_{0.1}$ represents the only object in membrane 0.1 for the initial configuration. There is no concern for the parallel structure (or sequence) of all sub processes in each membrane process P_θ .

Any object o is encoded as an output action $O_o \triangleq \overline{\sigma} \langle x \rangle_{@ps(cur)}$. The encoded *here* operation directly uses O_o to produce an object, the encoded *in* operation sends the object name to the address of the parent membrane node through an private system channel ch_θ , and the encoded *out* operation sends the object name to the address of the address of the grandfather membrane node through an public system channel ch .

Unconventionally while execution, we require that all possible reductions of the encoded process to be executed in parallel for each computation step. One can validate that the above process P_Π can simulate the computation of the corresponding P system Π .

5 Conclusion

This paper establishes a tight link between P systems and π calculus. Based upon the introduction of non-interleaving semantics, the membrane structure can be successfully encoded as a syntactic tree in π calculus. Although the encoded process in π calculus may decrease the computability, as π calculus has been widely used and implemented, it is still a stage to show the performance of P systems.

Due to page limits, context-sensitive rules are not discussed here. The idea to encode those rules is that using tuple match instead of single input/output. In short, $[(a, a, c) = (x, y, z)]P$ can be used to test whether we have objects aac . The technical development is rather lengthy.

The merit, someone regards as the drawback, of π calculus is that each single computation step can be traced so that the evolution can be precisely figured out while P systems emphasize in powerful concurrent computability.

References

- [1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.
- [2] A. V. Baranda, J. Castellanos, F. Arroyo, and R. Gonzalo. Towards an electronic implementation of membrane computing: A formal description of non-deterministic evolution in transition p systems. In N. Jonoska and N. Seeman, editors, *Proc. 7th Intern. Meeting on DNA Based Computers*, volume 2340 of *Lecture Notes in Computer Science*, pages 350–359, Tampa, Florida, USA, 2001.
- [3] C. Bodei, P. Degano, R. Focardi, and C. Priami. Primitives for authentication in process algebras. *Theor. Comput. Sci.*, 283(2):271–304, 2002.
- [4] P. Degano and C. Priami. Non-interleaving semantics for mobile processes. *Theor. Comput. Sci.*, 216(1-2):237–270, 1999.
- [5] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, (Parts I and II). *Information and Computation*, 100:1–77, 1992.
- [6] M. Pérez-Jiménez and F. Sancho-Caparrini. A formalization of basic p systems. *Fundamenta Informaticae*, 49(1-3):261–272, January 2002.
- [7] I. Petre and L. Petre. Mobile ambients and p-systems. *Journal of Universal Computer Science*, 5(9):588–598, Sept. 1999.
- [8] G. Păun. Computing with membranes: A variant. Technical report, 1999.
- [9] G. Păun. Computing with membranes. *J. Comput. Syst. Sci.*, 61(1):108–143, 2000.
- [10] G. Păun and G. Rozenberg. A guide to membrane computing. *Theoretical Computer Science*, 287(1):73–100, September 2002.
- [11] Z. Qi, J. You, and H. Mao. P Systems and Petri Nets. In *Proc. WMC 2003*, Lecture Notes in Computer Science.
- [12] P. Sewell. Applied pi - a brief tutorial. Technical report, 2000.