On the Expressiveness Power of Membrane Systems Working in Accepting Mode

Roberto Barbuti ¹, Andrea Maggiolo Schettini ¹, Paolo Milazzo ¹, Simone Tini ²

¹Università di Pisa, Pisa, Italy
²Università dell’Insubria, Como-Varese, Italy

Jena, August 25th, 2010
The expressiveness of several classes of P Systems viewed as generators of multisets is well known.

Our aim is to study the expressiveness of some classes of P Systems viewed as acceptors of multisets.
Definition

A membrane system $\Pi$ is given by

$$\Pi = (V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n)$$

where:

- $V$ is an alphabet whose elements are called objects;
- $\mu \subseteq \mathbb{N} \times \mathbb{N}$ is a membrane structure, such that $(l_1, l_2) \in \mu$ denotes that the membrane labeled by $l_2$ is contained in the membrane labeled by $l_1$;
- $w_j$ with $1 \leq j \leq n$ are strings from $V^*$ representing multisets over $V$ associated with the membranes $1, \ldots, n$ of $\mu$;
- $R_j$ with $1 \leq j \leq n$ are finite sets of evolution rules associated with the membranes $1, \ldots, n$ of $\mu$. 

R. Barbuti, A. Maggiolo Schettini, P. Milazzo, S. Tini
Expressiveness of Acceptor Membrane Systems
Assume $V$ partitioned into $\Sigma$ and the set of control objects $C$.

**Theorem**

*Every P system with promoters can be mapped to an equivalent flat (i.e. with a single membrane) P system.*


Here equivalent means that the two P Systems compute by having, step by step, the same multisets over $\Sigma$ in the skin membrane.
Definition

A flat generator over $\Sigma$ is a $P$ system

$$\Pi = (\Sigma \cup C, \emptyset, w_1, R_1)$$

A multiset of objects $w$ over $\Sigma$ is generated by $\Pi$ iff there exists a multiset $w'$ over $C$ and a final configuration that can be reached having $w \cup w'$ as multiset of objects.

Definition

A flat acceptor over $\Sigma$ is a $P$ system

$$\Pi = (\Sigma \cup C \cup \{T\}, \emptyset, w_1, R_1)$$

A multiset of objects $w$ over $\Sigma$ is accepted by $\Pi$ iff by adding $w$ to $w_1$ and by starting the computation, a final configuration can be reached with $T$ appearing in the membrane.
8 classes of P Systems

\[ P(\text{coo}/\text{ncoo}, \text{nondet}/\text{det}, \text{pro}/\text{npro}): \]
- \text{coo}: cooperative rules; \text{ncoo}: no cooperative rules.
- \text{nondet}: nondeterminism; \text{det}: determinism.
- \text{pro}: promoters, \text{npro}: no promoters.

16 classes of languages

\[ P_sP_x(\text{coo}/\text{ncoo}, \text{nondet}/\text{det}, \text{pro}/\text{nproo}): \]
- \( P_s \) stays for "Parikh set"
- \( x \) is "g" for generators and "a" for acceptors.
Assume a class of P Systems $C$ accepting/generating a class of languages $\mathcal{L}$.
Assume a class of P Systems $C'$ accepting/generating a class of languages $\mathcal{L}'$.

We write $\mathcal{L} \Rightarrow \mathcal{L}'$ iff there exists an encoding from $C$ to $C'$, namely, given any $\Pi \in C$ accepting/generating a language $L$, we can map it to some $\Pi' \in C'$ accepting/generating $L$. This implies that (but, in general, is not equivalent to) $\mathcal{L} \subseteq \mathcal{L}'$

We write $\mathcal{L} \Leftrightarrow \mathcal{L}'$ iff both $\mathcal{L} \Rightarrow \mathcal{L}'$ and $\mathcal{L}' \Rightarrow \mathcal{L}$.
\[ PsP_a(coo, ndet, pro) \parallel \uparrow \quad PsP_a(coo, ndet, npro) \quad PsP_a(ncoo, ndet, pro) \parallel \uparrow \]

\[ PsRE =* PsP_g(coo, ndet, pro) =* PsP_g(coo, ndet, npro) =* PsP_g(ncoo, ndet, npro) \parallel \uparrow \]

\[ PsRE = PsP_a(coo, det, pro) \iff PsP_a(coo, det, npro) \cup PsP_a(ncoo, det, pro) \cup PsP_a(ncoo, det, npro) \]

\[ \mathcal{L}_2 = PsP_g(coo, det, pro) = PsP_g(coo, det, npro) = PsP_g(ncoo, det, pro) = PsP_g(ncoo, det, npro) \]

where:

- \( \mathcal{L}_1 = \{ w \mid \exists A, N. w \cap A \neq \emptyset \land w \cap N = \emptyset \} \cup \{ w \mid \exists N. w \cap N = \emptyset \} \cup \{ \emptyset \} \)
- \( \mathcal{L}_2 = \{ \{ w \} \mid w \text{ is a multiset} \} \cup \{ \emptyset \} \)
- \( \mathcal{L}_3 \) is the least set containing \( \{ a^n \mid \exists k. n \geq k \} \) closed w.r.t. complementation, finite union and finite intersection.
Let us give an hint of some of these results.
\[ \begin{align*}
PsP_a(\text{coo}, \text{ndet}, \text{pro}) & \parallel \uparrow \quad PsP_a(\text{coo}, \text{ndet}, \text{npro}) \\
PsRE & = \ast PsP_g(\text{coo}, \text{ndet}, \text{pro}) = \ast PsP_g(\text{coo}, \text{ndet}, \text{npro}) = \ast PsP_g(\text{ncoo}, \text{ndet}, \text{pro}) \\
\end{align*} \]
We map a generator $\Pi = (\Sigma \cup C, \emptyset, w, R) \in P(ncoo, ndet, pro)$ into an equivalent acceptor $\Pi_a = (\Sigma_a \cup C_a \cup \{T\}, \emptyset, w_a, R_a)$. Idea: Rename all objects $a$ of $\Pi$ as $a'$, embed the $\Pi$ so modified into $\Pi_a$, generate a multiset and check if it coincides with the input of $\Pi_a$:

1. Given any input $u$, use $\Pi$ to generate a multiset $v'$.
2. When $\Pi$ has terminated, start the comparison between $u$ and $v'$.
3. If $u = v'$ then accept $u$. The nondeterminism ensures that all multisets generated by $\Pi$ can be considered.

Observation: we have to check when $\Pi$ has terminated and we have to start the comparison.
Embed $\Pi$ into $\Pi_a$ by renaming all $a$ in $\Pi$ as $a'$:

$$w_a \supseteq w' \text{ and } R_a \supseteq \{a' \rightarrow v'g | \rho' \text{ s. t. } a \rightarrow v|\rho \text{ is a rule in } R\}$$

$$R_a \supseteq \{g \rightarrow \lambda\}$$

In this way, $\Pi_a$ generates a multiset $u'$ iff $\Pi$ generates $u$.

Now, if the input to $\Pi_a$ is $u$ then $\Pi_a$ should accept it, since $\Pi_a$ should accept exactly the multisets generated by $\Pi$.

Object $g$ means that $\Pi$ is still working.

So, we have to check if the input and the multiset generated by $\Pi$ are the same. Such a checking will be enabled only when $\Pi$ terminates, i.e. when $g$ disappears.
1 \{a' \rightarrow v'g_{p'}, \text{s. t. } a \rightarrow v_{p} \text{ is a rule in } R\} \cup \{g \rightarrow \lambda\}

2 \text{Add multiset } x1g \text{ to } \omega_{a} \text{ and rules: }

\begin{align*}
    x & \rightarrow x'_{1}g \\
    x' & \rightarrow x_{2} \\
    x & \rightarrow s_{2} \\
    1 & \rightarrow 2 \\
    2 & \rightarrow 1_{x'}
\end{align*}

Until \( \Pi \) is working, object \( g \) is generated by rules added at Item 1.

When \( \Pi \) terminates, \( g \) is no more generated, and \( s \) is produced.

Object \( s \) triggers the comparison between the input and the object generated by \( \Pi \).
Add objects $\bar{0}$ and $T$ to $w_a$ and rules:

1. \[ \{ a \to a_{\bar{1}} \mid a \in \Sigma \} \]
2. \[ \{ a \to A_{\bar{1}} \mid a \in \Sigma \} \]
3. \[ \{ a' \to a'_{\bar{1}} \mid a \in \Sigma \} \]
4. \[ \{ a' \to A'_{\bar{1}} \mid a \in \Sigma \} \]
5. \[ \{ T \to \lambda_{AB_{\bar{2}}} \mid A, B \in \hat{C} \} \]
6. \[ \{ T \to \lambda_{A'B'_{\bar{2}}} \mid A', B' \in \hat{C} \} \]
7. \[ \{ T \to \lambda_{A_{\bar{3}}} \mid A \in \hat{C} \} \]
8. \[ \{ T \to \lambda_{A'_{\bar{3}}} \mid A' \in \hat{C} \} \]
9. \[ \{ T \to T_{AA'_{\bar{3}}} \mid A, A' \in \hat{C} \} \]
10. \[ \{ A \to \lambda_{\bar{3}} \mid A \in \hat{C} \} \]
11. \[ \{ A' \to \lambda_{\bar{3}} \mid A' \in \hat{C} \} \]
\[
\begin{align*}
\text{PsP}_a(c\text{oo}, \text{ndet}, \text{pro}) & \parallel \uparrow \text{PsP}_a(c\text{oo}, \text{ndet}, \text{npro}) \\
\text{PsRE} = \ast \text{PsP}_g(c\text{oo}, \text{ndet}, \text{pro}) & = \ast \text{PsP}_g(c\text{oo}, \text{ndet}, \text{npro}) = \ast \text{PsP}_a(n\text{coo}, \text{ndet}, \text{pro}) \parallel \uparrow \text{PsP}_a(n\text{coo}, \text{ndet}, \text{npro}) \\
\text{PsRE} = \text{PsP}_a(c\text{oo}, \text{det}, \text{pro}) \Leftrightarrow \text{PsP}_a(c\text{oo}, \text{det}, \text{npro}) \cup \text{PsP}_a(n\text{coo}, \text{det}, \text{pro}) \cup \text{PsP}_a(n\text{coo}, \text{det}, \text{npro}) \\
\text{PsRE} = \text{PsP}_g(c\text{oo}, \text{det}, \text{pro}) = \text{PsP}_g(c\text{oo}, \text{det}, \text{npro}) = \text{PsP}_g(n\text{coo}, \text{det}, \text{pro}) = \text{PsP}_g(n\text{coo}, \text{det}, \text{npro}) \\
\text{L}_2 = \text{PsP}_g(c\text{oo}, \text{det}, \text{pro}) = \text{PsP}_g(c\text{oo}, \text{det}, \text{npro}) = \text{PsP}_g(n\text{coo}, \text{det}, \text{pro}) = \text{PsP}_g(n\text{coo}, \text{det}, \text{npro}) \\
\text{L}_3 \cup \text{PsP}_a(c\text{oo}, \text{det}, \text{pro}) \parallel \text{PsP}_a(c\text{oo}, \text{det}, \text{npro}) \cup \text{PsP}_a(n\text{coo}, \text{det}, \text{pro}) \cup \text{PsP}_a(n\text{coo}, \text{det}, \text{npro}) \cup \text{PsP}_g(c\text{oo}, \text{det}, \text{pro}) \cup \text{PsP}_g(c\text{oo}, \text{det}, \text{npro}) \cup \text{PsP}_g(n\text{coo}, \text{det}, \text{pro}) \cup \text{PsP}_g(n\text{coo}, \text{det}, \text{npro})
\end{align*}
\]
As in the case of $P(ncoo, ndet, pro)$ we embed a generator into an acceptor and we compare the input with a generated multiset. Items 1 and 2 are as in the previous case, the comparison (i.e. Item 3) is simpler due to cooperative rules:

1. \[
\left\{ u' \rightarrow v' g | p' \right\} \text{ s.t. } u \rightarrow_p v \text{ is a rule in } R \right\} \cup \{ g \rightarrow \lambda \}
\]

2. \[
x \rightarrow x' |_1 g \quad x' \rightarrow x |_2 \quad x \rightarrow s |_2 \quad 1 \rightarrow 2 \quad 2 \rightarrow 1 |_{x'}
\]

3. Add object $T$ to $w_a$ and cooperative rules:

\[
\left\{ a a' \rightarrow \lambda |_s \text{ s.t. } a \in \Sigma \right\}
\]
\[
\left\{ aT \rightarrow \lambda |_s \text{ s.t. } a \in \Sigma \right\}
\]
\[
\left\{ a' T \rightarrow \lambda |_s \text{ s.t. } a \in \Sigma \right\}
\]
\[
P_{S^P_a}(\text{coo, ndet, pro}) \parallel \uparrow \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, pro}) \quad \equiv \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, npro}) \\
\]} \quad \]

\[
P_{S^P_a}(\text{coo, ndet, pro}) \parallel \uparrow \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, pro}) \quad \equiv \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, npro}) \\
\]} \quad \]

\[
P_{S^P_a}(\text{coo, ndet, pro}) \parallel \uparrow \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, pro}) \quad \equiv \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, npro}) \\
\]} \quad \]

\[
P_{S^P_a}(\text{coo, ndet, pro}) \parallel \uparrow \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, pro}) \quad \equiv \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, npro}) \\
\]} \quad \]

\[
P_{S^P_a}(\text{coo, ndet, pro}) \parallel \uparrow \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, pro}) \quad \equiv \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, npro}) \\
\]} \quad \]

\[
P_{S^P_a}(\text{coo, ndet, pro}) \parallel \uparrow \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, pro}) \quad \equiv \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, npro}) \\
\]} \quad \]

\[
P_{S^P_a}(\text{coo, ndet, pro}) \parallel \uparrow \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, pro}) \quad \equiv \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, npro}) \\
\]} \quad \]

\[
P_{S^P_a}(\text{coo, ndet, pro}) \parallel \uparrow \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, pro}) \quad \equiv \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, npro}) \\
\]} \quad \]

\[
P_{S^P_a}(\text{coo, ndet, pro}) \parallel \uparrow \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, pro}) \quad \equiv \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, npro}) \\
\]} \quad \]

\[
P_{S^P_a}(\text{coo, ndet, pro}) \parallel \uparrow \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, pro}) \quad \equiv \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, npro}) \\
\]} \quad \]

\[
P_{S^P_a}(\text{coo, ndet, pro}) \parallel \uparrow \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, pro}) \quad \equiv \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, npro}) \\
\]} \quad \]

\[
P_{S^P_a}(\text{coo, ndet, pro}) \parallel \uparrow \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, pro}) \quad \equiv \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, npro}) \\
\]} \quad \]

\[
P_{S^P_a}(\text{coo, ndet, pro}) \parallel \uparrow \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, pro}) \quad \equiv \quad \text{PsRE} =^* P_{S^P_g}(\text{coo, ndet, npro}) \\
Also in this case we embed the generator \( \Pi = (\Sigma \cup C, \emptyset, w, R) \) into an equivalent acceptor \( \Pi_a = (\Sigma_a \cup C_a \cup \{T\}, \emptyset, w_a, R_a) \).

- \( w' \subseteq w_a, \ t \in w_a \) and \( T \in w_a \).
- \( t \rightarrow rs \in R_a \). Object \( s \) triggers the comparison.
- The work by \( \Pi \) is simulated by a loop with 3 steps:
  1. \( R_a \supseteq \{u' \rightarrow v''v''', tu' \rightarrow v''v'''t' \text{ s.t. } u \rightarrow v \text{ is a rule in } R\} \)
  2. \( R_a \supseteq \{a''''rT \rightarrow \lambda \text{ s.t. } a \in \Sigma\} \cup \{t' \rightarrow t''\} \cup \{a'' \rightarrow a''' \mid a \in \Sigma\} \)
  3. \( R_a \supseteq \{a''''a''''' \rightarrow a' \text{ s.t. } a \in \Sigma\} \cup \{t'' \rightarrow t\} \)
- If \( t \rightarrow rs \) fires before the loop terminates, rule \( a''''rT \rightarrow \lambda \) removes \( T \) and the computation is not accepting.
- Otherwise, after the loop the comparison starts:
  \( \{sa \rightarrow s \mid a \in \Sigma\} \cup \{saT \rightarrow \lambda \mid a \in \Sigma\} \cup \{sa'T \rightarrow \lambda \mid a \in \Sigma\} \)
\[ PsP_a(\text{coo}, \text{ndet}, \text{pro}) \quad \parallel \uparrow \quad PsP_a(\text{coo}, \text{ndet}, \text{npro}) \quad =^* \quad PsP_a(\text{ncoo}, \text{ndet}, \text{pro}) \]

\[ PsRE =^* PsP_g(\text{coo}, \text{ndet}, \text{pro}) =^* PsP_g(\text{coo}, \text{ndet}, \text{npro}) =^* PsP_g(\text{ncoo}, \text{ndet}, \text{npro}) \quad =^* \quad PsP_g(\text{ncoo}, \text{ndet}, \text{pro}) \]

\[ PsRE = PsP_a(\text{coo}, \text{det}, \text{pro}) \equiv PsP_a(\text{coo}, \text{det}, \text{npro}) \supset PsP_a(\text{ncoo}, \text{det}, \text{pro}) \supset PsP_a(\text{ncoo}, \text{det}, \text{npro}) \]

\[ L_1 = \{ w \mid \exists A, N. w \cap A \neq \emptyset \land w \cap N = \emptyset \} \cup \{ w \mid \exists N. w \cap N = \emptyset \} \cup \{ \emptyset \} \]

where \( \mathcal{L}_1 \) is defined as above.
An acceptor in $P(ncoo, det, npro)$ for $\{w \mid \exists A, N. w \cap A \neq \emptyset \land w \cap N = \emptyset\}$ has no control object and rules $\{a \to T \mid a \in A\}$ and $\{b \to b \mid b \in N\}$.

An acceptor in $P(ncoo, det, npro)$ for $\{w \mid \exists N. w \cap N = \emptyset\}$ contains initially an occurrence of $T$ and has rules $\{b \to b \mid b \in N\}$.

To see that $PsP_a(ncoo, ndet, npro) \subseteq L_1$, take any acceptor $\Pi \in P(ncoo, ndet, npro)$. If it contains a rule of the form $T \to u$, for any $u$, then $Ps(\Pi) = \emptyset$, and $\emptyset \in L_1$. Otherwise, let $G$ be the graph having a node for each object in $\Sigma \cup C$ and an arch from $a$ to $b$ if there is a rule $a \to u$ with $b \in u$.

Let $N$ be the set of the objects $a \in \Sigma$ such that all paths from $a$ are infinite, i.e. $a \to \cdots \to a' \to \cdots \to a'$ for some $a'$.

Let $A$ be the set of the objects $a \in \Sigma$ such that at least one path from $a$ is finite and leads to $T$, i.e. $a \to \cdots \to T$.

If $T$ is an initial object in $\Pi$ then a multiset is accepted iff it gives rise to a finite computation, because no rule can remove $T$ and the final configuration, if reached, contains $T$ for sure. Therefore, $Ps(\Pi) = \{w \mid w \cap N = \emptyset\}$.

If $T$ is not initially in $\Pi$, then a multiset is accepted iff it gives rise to a finite computation that introduces $T$ in one of its steps. Therefore, $Ps(\Pi) = \{w \mid w \cap A \neq \emptyset \land w \cap N = \emptyset\}$.  

R. Barbuti, A. Maggiolo Schettini, P. Milazzo, S. Tini  
Expressiveness of Acceptor Membrane Systems
\[ PsP_a(\text{coo, ndet, pro}) \parallel \uparrow \] 
\[ PsRE \equiv^* PsP_g(\text{coo, ndet, pro}) =^* PsP_g(\text{coo, ndet, npro}) =^* PsP_g(\text{ncoo, ndet, npro}) \]

\[ \bigcup \] 
\[ L_1 \] 
\[ \bigcup \] 
\[ L_3 \]

\[ PsRE \equiv PsP_a(\text{coo, det, pro}) \leftrightarrow PsP_a(\text{coo, det, npro}) \supset PsP_a(\text{ncoo, det, pro}) \supset PsP_a(\text{ncoo, det, npro}) \]

\[ \bigcup \] 
\[ L_2 \] 
\[ = PsP_g(\text{coo, det, pro}) = PsP_g(\text{coo, det, npro}) = PsP_g(\text{ncoo, det, pro}) = PsP_g(\text{ncoo, det, npro}) \]
We map a 3-register machine $M$ to an equivalent acceptor $\Pi_M$ in $P(\text{coo, det, npro})$. Let $R_1, R_2, R_3$ be the three registers of $M$, and $0 \leq i \leq m$ be the instructions. The idea is to represent a configuration $(i, A, B, C)$ with multiset $(ia^A b^B c^C)$.

Instruction $i : R_1 +, j$ is simulated by rule $i \rightarrow aj$.

Instruction $i : R_1 -, j, k$ is simulated by rules

$$i \rightarrow x_i y_i \quad ax_i \rightarrow x'_i \quad y_i \rightarrow y'_i \quad y'_i x'_i \rightarrow j \quad y'_i x_i \rightarrow k.$$

Instructions over $R_2$ and $R_3$ are analogous, we simply replace any occurrence of $a$ with $b$ or $c$, respectively.

Finally, we need these rules to check that configuration $(0, 0, 0, 0)$ has been reached:

$$0 \rightarrow T \quad Ta \rightarrow \lambda \quad Tb \rightarrow \lambda \quad Tc \rightarrow \lambda.$$
\[ PsP_a(\text{coo, ndet, pro}) \uparrow \parallel \uparrow PsP_g(\text{coo, ndet, pro}) \]

\[ PsRE \; \; \; =^* \; \; \; PsP_g(\text{coo, ndet, pro}) \]

\[ PsP_a(\text{coo, ndet, npro}) \parallel \uparrow \]

\[ PsP_g(\text{coo, ndet, npro}) \]

\[ PsP_a(\text{ncoo, ndet, pro}) \parallel \uparrow \]

\[ PsP_g(\text{ncoo, ndet, npro}) \]

\[ \cup^* \]

\[ L_1 \]

\[ \cup \]

\[ L_3 \]

\[ PsRE \; \; \; = PsP_a(\text{coo, det, pro}) \rightleftharpoons PsP_a(\text{coo, det, npro}) \cup PsP_a(\text{ncoo, det, pro}) \cup PsP_a(\text{ncoo, det, npro}) \]

\[ L_2 \; \; \; = PsP_g(\text{coo, det, pro}) = PsP_g(\text{coo, det, npro}) = PsP_g(\text{ncoo, det, pro}) = PsP_g(\text{ncoo, det, npro}) \]
Take any \( \Pi = (\Sigma \cup C, \emptyset, w, R) \in P(\text{coo, det, pro}) \) with \( R = \{r_1, \ldots, r_k\} \).

Idea: rewrite any \( r_i \equiv u_i \rightarrow v_i|p_i \) as \( r'_i \equiv u_ip_i \rightarrow v_ip_i \).

This requires that \( u_i \cap p_i = \emptyset \): if not, rewrite first \( r_i \) as \( u_i \rightarrow v_i|p_i \setminus u_i \).

By moving promoters to left hand sides of rules we may introduce nondeterminism. (For example, by transforming rules \( a \rightarrow d|c \) and \( b \rightarrow e|c \) into \( ac \rightarrow dc \) and \( bc \rightarrow ec \).

So, rewrite \( r'_i \) as \( iu_ip_i \rightarrow v_ip_i \), where \( 1 \leq i \leq k \) are new control objects that must be introduced in sequence.

If \( v_i \cap u_i \neq \emptyset \) then performing \( r'_i \) may trigger \( r'_i \) itself. So, rewrite \( r'_i \) as \( iu'_ip_i \rightarrow v'_ip'_i \).

We may run \( r'_i \) more than once: rewrite it as \( i'i''p'_i \rightarrow v''p''i''' \) and add rules:

\[
\{ i \rightarrow i'i'', i'' \rightarrow i''', i''''i''' \rightarrow i, i'i''' \rightarrow i + 1 \}
\]

So, we simulate an original computation step by first running \( r'_1 \), then \( r'_2 \), and so on.

We also require rules for:

- Map all objects \( a \rightarrow a' \) before object 1 is introduced.
- Map all \( a' \) not consumed by rules and all \( a'' \) introduced by rules to \( a \).