# On the Expressiveness Power of Membrane Systems Working in Accepting Mode

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

[1] Università di Pisa, Pisa, Italy
[2] Università 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*;
- $\mu \subset N \times N$ 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 $\mu$.

Assume $V$ partitioned into $\Sigma$ and the set of *control objects* $\mathcal{C}$.

**Theorem**

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

- Z. Qi, J. You, H. Mao, WMC 2003.
- L. Bianco, V. Manca, WMC 2005.
- R. Barbuti, A. Maggiolo Schettini, P. Milazzo, S. Tini, Fundamenta Informaticae 87, 2008.

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

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

A multiset of objects $w$ over $\Sigma$ is generated by $\Pi$ iff there exists a multiset $w'$ over $\mathcal{C}$ 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

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

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

$P(coo/ncoo, ndet/det, pro/npro)$:

- *coo*: cooperative rules; *ncoo*: no cooperative rules.
- *ndet*: nondeterminism; *det*: determinism.
- *pro*: promoters, *npro*: no promoters.

16 classes of languages

$P_s P_x(coo/ncoo, ndet/det, pro/nproo)$:

- $P_s$ 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 $\mathcal{L}' \Rightarrow \mathcal{L}$.

$$
\begin{array}{ccccccccc}
 & & PsP_a(coo, ndet, pro) & & PsP_a(coo, ndet, npro) & & & & PsP_a(ncoo, ndet, pro) \\
 & & \| \Uparrow & & \| \Uparrow & & & & \| \Uparrow \\
PsRE & =^* & PsP_g(coo, ndet, pro) & =^* & PsP_g(coo, ndet, npro) & & & =^* & PsP_g(ncoo, ndet, pro) \\
 & & & & & & & & \cup^* \\
 & & & & & & & & PsP_g(ncoo, ndet, npro) \\
 & & & & & & & & \cup \\
 & & & & & & & & \mathcal{L}_1 \\
 & & & & & & & & \| \\
 & & & & & & \mathcal{L}_3 & & PsP_a(ncoo, ndet, npro) \\
 & & & & & & \| & & \| \\
PsRE & = & PsP_a(coo, det, pro) & \overset{\Leftrightarrow}{\subseteq} & PsP_a(coo, det, npro) & \supset & PsP_a(ncoo, det, pro) & \supset & PsP_a(ncoo, det, npro) \\
 & & \cup & & \cup & & \cup & & \cup \\
\mathcal{L}_2 & = & PsP_g(coo, det, pro) & = & PsP_g(coo, det, npro) & = & PsP_g(ncoo, det, pro) & = & PsP_g(ncoo, det, npro)
\end{array}
$$

where:

- $\mathcal{L}_1 = \{w \mid \exists A, N.\, w \cap A \neq \emptyset \wedge w \cap N = \emptyset\} \cup \{w \mid \exists N.\, w \cap N = \emptyset\} \cup \{\varnothing\}$
- $\mathcal{L}_2 = \{\{w\} \mid w \text{ is a multiset}\} \cup \{\varnothing\}$
- $\mathcal{L}_3$ 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.

$$
\begin{array}{lccccccc}
 & PsP_a(coo, ndet, pro) & & PsP_a(coo, ndet, npro) & & & & PsP_a(ncoo, ndet, pro) \\
 & \| \Uparrow & & \| \Uparrow & & & & \| \Uparrow \\
PsRE & =^{*} PsP_g(coo, ndet, pro) & =^{*} & PsP_g(coo, ndet, npro) & & =^{*} & & PsP_g(ncoo, ndet, pro) \\
 & & & & & & & \cup^{*} \\
 & & & & & & & PsP_g(ncoo, ndet, npro) \\
 & & & & & & & \cup \\
 & & & & & & & \mathcal{L}_1 \\
 & & & & & & & \| \\
 & & & & \mathcal{L}_3 & & & PsP_a(ncoo, ndet, npro) \\
 & & & & \| & & & \| \\
PsRE & = PsP_a(coo, det, pro) & \overset{\Leftrightarrow}{\underset{=}{?}} & PsP_a(coo, det, npro) & \supset PsP_a(ncoo, det, pro) & \supset & & PsP_a(ncoo, det, npro) \\
 & \cup & & \cup & \cup & & & \cup \\
\mathcal{L}_2 & = PsP_g(coo, det, pro) & = & PsP_g(coo, det, npro) & = PsP_g(ncoo, det, pro) & = & & PsP_g(ncoo, det, npro)
\end{array}
$$

We map a generator $\Pi = (\Sigma \cup \mathcal{C}, \emptyset, w, R) \in P(ncoo, ndet, pro)$ into an equivalent acceptor $\Pi_a = (\Sigma_a \cup \mathcal{C}_a \cup \{T\}, \emptyset, w_a, R_a)$.
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.

1. Embed $\Pi$ into $\Pi_a$ by renaming all $a$ in $\Pi$ as $a'$:

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

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

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. $\{a' \rightarrow v'g|_{p'}$ s. t. $a \rightarrow v|_p$ is a rule in $R\} \cup \{g \rightarrow \lambda\}$
2. Add multiset $x1g$ to $w_a$ and rules:

$$x \rightarrow x'|_{1g} \quad x' \rightarrow x|_2 \quad x \rightarrow s|_2 \quad 1 \rightarrow 2 \quad 2 \rightarrow 1|_{x'}$$

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 $\Pi$.

1. $\{a' \to v'g|_{p'}$ s. t. $a \to_p v$ is a rule in $R\}$
2. $x \to x'|_{1g} \quad x' \to x|_2 \quad x \to s|_2 \quad g \to \lambda \quad 1 \to 2 \quad 2 \to \ |_{x'}$
3. Add objects $\overline{0}$ and $T$ to $w_a$ and rules:

$$\overline{0} \to \overline{1}|_s \qquad \overline{1} \to \overline{2} \qquad \overline{2} \to \overline{3} \qquad \{\overline{3} \to \overline{1}|_{Taa'} \mid a \in \Sigma, a' \in \Sigma'\}$$

$$\{ a \to a|_{\overline{1}} \mid a \in \Sigma \} \qquad \{ a \to A|_{\overline{1}} \mid a \in \Sigma \}$$
$$\{ a' \to a'|_{\overline{1}} \mid a \in \Sigma \} \qquad \{ a' \to A'|_{\overline{1}} \mid a \in \Sigma \}$$

$$\{ T \to \lambda|_{AB\overline{2}} \mid A, B \in \hat{C} \} \qquad \{ T \to \lambda|_{A'B'\overline{2}} \mid A', B' \in \hat{C} \}$$

$$\{ T \to \lambda|_{A\overline{3}} \mid A \in \hat{C} \} \qquad \{ T \to \lambda|_{A'\overline{3}} \mid A' \in \hat{C} \}$$
$$\{ T \to T|_{AA'\overline{3}} \mid A, A' \in \hat{C} \}$$
$$\{ A \to \lambda|_{\overline{3}} \mid A \in \hat{C} \} \qquad \{ A' \to \lambda|_{\overline{3}} \mid A' \in \hat{C} \}$$

$$
\begin{array}{ccccccc}
 & & \color{red}{PsP_a(coo, ndet, pro)} & & PsP_a(coo, ndet, npro) & & PsP_a(ncoo, ndet, pro) \\
 & & \| \; \color{red}{\Uparrow} & & \| \; \Uparrow & & \| \; \Uparrow \\
PsRE & =^* & \color{red}{PsP_g(coo, ndet, pro)} & =^* & PsP_g(coo, ndet, npro) & =^* & PsP_g(ncoo, ndet, pro) \\
 & & & & & & \cup^* \\
 & & & & & & PsP_g(ncoo, ndet, npro) \\
 & & & & & & \cup \\
 & & & & & & \mathcal{L}_1 \\
 & & & & & & \| \\
 & & & & \mathcal{L}_3 & & PsP_a(ncoo, ndet, npro) \\
 & & & & \| & & \| \\
PsRE & = PsP_a(coo, det, pro) & \overset{\Leftrightarrow}{\subseteq} & PsP_a(coo, det, npro) \supset & PsP_a(ncoo, det, pro) & \supset & PsP_a(ncoo, det, npro) \\
 & \cup & & \cup & \cup & & \cup \\
\mathcal{L}_2 & = PsP_g(coo, det, pro) & = & PsP_g(coo, det, npro) = & PsP_g(ncoo, det, pro) & = & PsP_g(ncoo, det, npro)
\end{array}
$$

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. $\{u' \to v'g|_{p'}$ s. t. $u \to_p v$ is a rule in $R\} \cup \{g \to \lambda\}$
2. $x \to x'|_{1g} \quad x' \to x|_2 \quad x \to s|_2 \quad 1 \to 2 \quad 2 \to 1|_{x'}$
3. Add object $T$ to $w_a$ and cooperative rules:

$$\{aa' \to \lambda|_s \text{ s.t. } a \in \Sigma\}$$
$$\{aT \to \lambda|_s \text{ s.t. } a \in \Sigma\}$$
$$\{a'T \to \lambda|_s \text{ s.t. } a \in \Sigma\}$$

$$
\begin{array}{lcccccc}
 & & PsP_a(coo, ndet, pro) & & PsP_a(coo, ndet, npro) & & PsP_a(ncoo, ndet, pro) \\
 & & \| \Uparrow & & \| \Uparrow & & \| \Uparrow \\
PsRE & =^* & PsP_g(coo, ndet, pro) & =^* & PsP_g(coo, ndet, npro) & =^* & PsP_g(ncoo, ndet, pro) \\
 & & & & & & \cup^* \\
 & & & & & & PsP_g(ncoo, ndet, npro) \\
 & & & & & & \cup \\
 & & & & & & \mathcal{L}_1 \\
 & & & & & & \| \\
 & & & & & \mathcal{L}_3 & PsP_a(ncoo, ndet, npro) \\
 & & & & & \| & \| \\
PsRE & = & PsP_a(coo, det, pro) \stackrel{\Leftrightarrow}{=} & PsP_a(coo, det, npro) & \supset & PsP_a(ncoo, det, pro) \supset & PsP_a(ncoo, det, npro) \\
 & & \cup & \cup & & \cup & \cup \\
\mathcal{L}_2 & = & PsP_g(coo, det, pro) = & PsP_g(coo, det, npro) & = & PsP_g(ncoo, det, pro) = & PsP_g(ncoo, det, npro)
\end{array}
$$

Also in this case we embed the generator $\Pi = (\Sigma \cup \mathcal{C}, \emptyset, w, R)$ into an equivalent acceptor $\Pi_a = (\Sigma_a \cup \mathcal{C}_a \cup \{T\}, \emptyset, w_a, R_a)$.

- $w' \subseteq w_a$, $t \in w_a$ 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. $R_a \supseteq \{u' \rightarrow v''v''', tu' \rightarrow v''v'''t'$ s. t. $u \rightarrow v$ is a rule in $R\}$
  2. $R_a \supseteq \{a'''rT \rightarrow \lambda$ s.t. $a \in \Sigma\} \cup \{t' \rightarrow t''\} \cup \{a'' \rightarrow a'''' \mid a \in \Sigma\}$
  3. $R_a \supseteq \{a'''a'''' \rightarrow a'$ 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: $\{saa' \rightarrow s \mid a \in \Sigma\} \cup \{saT \rightarrow \lambda \mid a \in \Sigma\} \cup \{sa'T \rightarrow \lambda \mid a \in \Sigma\}$

$$\begin{array}{ccccccc}
 & & PsP_a(coo, ndet, pro) & & PsP_a(coo, ndet, npro) & & PsP_a(ncoo, ndet, pro) \\
 & & \| \Uparrow & & \| \Uparrow & & \| \Uparrow \\
PsRE & =^* & PsP_g(coo, ndet, pro) & =^* & PsP_g(coo, ndet, npro) & =^* & PsP_g(ncoo, ndet, pro) \\
 & & & & & & \cup^* \\
 & & & & & & PsP_g(ncoo, ndet, npro) \\
 & & & & & & \cup \\
 & & & & & & {\color{red}\mathcal{L}_1} \\
 & & & & & & {\color{red}\|}
\end{array}$$

$$\begin{array}{ccccccccc}
 & & & & & & \mathcal{L}_3 & & {\color{red}PsP_a(ncoo, ndet, npro)} \\
 & & & & & & \| & & {\color{red}\|} \\
PsRE & = & PsP_a(coo, det, pro) & \overset{\Leftrightarrow}{\leqq} & PsP_a(coo, det, npro) & \supset & PsP_a(ncoo, det, pro) & \supset & {\color{red}PsP_a(ncoo, det, npro)} \\
 & & \cup & & \cup & & \cup & & \cup \\
\mathcal{L}_2 & = & PsP_g(coo, det, pro) & = & PsP_g(coo, det, npro) & = & PsP_g(ncoo, det, pro) & = & PsP_g(ncoo, det, npro)
\end{array}$$

where $\mathcal{L}_1 = \{w \mid \exists A, N.\ w \cap A \neq \emptyset \wedge w \cap N = \emptyset\} \cup \{w \mid \exists N.\ w \cap N = \emptyset\} \cup \{\varnothing\}$

- An acceptor in $P(ncoo, det, npro)$ for $\{w \mid \exists A, N.\ w \cap A \neq \emptyset \wedge w \cap N = \emptyset\}$ 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 $PsP_a(ncoo, ndet, npro) \subseteq \mathcal{L}_1$, 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 $\emptyset \in \mathcal{L}_1$.
  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, $Ps(\Pi) = \{w \mid w \cap A \neq \emptyset \wedge w \cap N = \emptyset\}$.

$$
\begin{array}{lcccc}
 & PsP_a(coo, ndet, pro) & & PsP_a(coo, ndet, npro) & & & PsP_a(ncoo, ndet, pro) \\
 & \| \Uparrow & & \| \Uparrow & & & \| \Uparrow \\
PsRE & =^* PsP_g(coo, ndet, pro) & =^* & PsP_g(coo, ndet, npro) & & =^* & PsP_g(ncoo, ndet, pro) \\
 & & & & & & \cup^* \\
 & & & & & & PsP_g(ncoo, ndet, npro) \\
 & & & & & & \cup \\
 & & & & & & \mathcal{L}_1 \\
 & & & & & & \| \\
 & & & & \mathcal{L}_3 & & PsP_a(ncoo, ndet, npro) \\
 & & & & \| & & \| \\
\color{red}{PsRE} & \color{red}{= PsP_a(coo, det, pro)} & \overset{\Leftrightarrow}{\subseteq} & PsP_a(coo, det, npro) & \supset PsP_a(ncoo, det, pro) & \supset & PsP_a(ncoo, det, npro) \\
 & \cup & & \cup & \cup & & \cup \\
\mathcal{L}_2 & = PsP_g(coo, det, pro) & = & PsP_g(coo, det, npro) & = PsP_g(ncoo, det, pro) & = & PsP_g(ncoo, det, npro)
\end{array}
$$

We map a 3-register machine $M$ to an equivalent acceptor $\Pi_M$ in $P(coo, det, npro)$.

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 $i \to aj$.

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

$$i \to x_i y_i \qquad ax_i \to x_i' \qquad y_i \to y_i' \qquad y_i' x_i' \to j \qquad y_i' x_i \to k\,.$$

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:

$$0 \to T \qquad Ta \to \lambda \qquad Tb \to \lambda \qquad Tc \to \lambda\,.$$

$$
\begin{array}{ccccccc}
 & PsP_a(coo, ndet, pro) & & PsP_a(coo, ndet, npro) & & & PsP_a(ncoo, ndet, pro) \\
 & \| \Uparrow & & \| \Uparrow & & & \| \Uparrow \\
PsRE =^{*} & PsP_g(coo, ndet, pro) & =^{*} & PsP_g(coo, ndet, npro) & =^{*} & & PsP_g(ncoo, ndet, pro) \\
 & & & & & & \cup^{*} \\
 & & & & & & PsP_g(ncoo, ndet, npro) \\
 & & & & & & \cup \\
 & & & & & & \mathcal{L}_1 \\
 & & & & & & \| \\
 & & & & & \mathcal{L}_3 & PsP_a(ncoo, ndet, npro) \\
 & & & & & \| & \| \\
PsRE = & PsP_a(coo, det, pro) & \stackrel{\Leftrightarrow}{=} & PsP_a(coo, det, npro) & \supset & PsP_a(ncoo, det, pro) \supset & PsP_a(ncoo, det, npro) \\
 & \cup & & \cup & & \cup & \cup \\
\mathcal{L}_2 = & PsP_g(coo, det, pro) & = & PsP_g(coo, det, npro) & = & PsP_g(ncoo, det, pro) = & PsP_g(ncoo, det, npro)
\end{array}
$$

Take any $\Pi = (\Sigma \cup C, \emptyset, w, R) \in P(coo, det, pro)$ with $R = \{r_1, \ldots, r_k\}$.

Idea: rewrite any $r_i \equiv u_i \rightarrow v_i|_{p_i}$ as $r_i' \equiv u_i p_i \rightarrow v_i p_i$.

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 trasforming rules $a \rightarrow d|_c$ and $b \rightarrow e|_c$ into $ac \rightarrow dc$ and $bc \rightarrow ec$.) So, rewrite $r_i'$ as $iu_i p_i \rightarrow v_i p_i$, 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 $iu_i' p_i' \rightarrow v_i'' p_i'$.

We may run $r_i'$ more than once: rewrite it as $i' u_i' p_i' \rightarrow v_i'' p_i' i'''$ 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 $a$ to $a'$ before object 1 is introduced.
- Map all $a'$ not consumed by rules and all $a''$ introduced by rules to $a$.