Compositional asynchronous membrane systems*

Cosmin Bonchiş[1], Cornel Izbaşa[1] and Gabriel Ciobanu[2]**

(1. Research Institute "e-Austria" Timişoara, Romania; 2. "A. I. Cuza" University, Faculty of Computer Science and Romanian Academy, Institute of Computer Science Blvd. Carol I no. 8, 700505 Iaşi, Romania)

Abstract This paper presents an algorithmic way of building complex membrane systems by coupling elementary membranes. Its application seems particularly valuable in the case of asynchronous membrane systems, since the resulting membrane system remains asynchronous. The composition method is based on a handshake mechanism implemented by using antiport rules and promoters.

Keywords: complex membrane systems, coupling elementary membranes, asynchronous membrane systems.

Membrane computing is inspired by the structure, functioning and organization of living cells. It aims to abstract computing ideas and models from living cells. An important feature in membrane computing is the processing of multisets of objects in a localized manner via evolution rules encapsulated in compartments. The rules are applied in a maximally parallel and nondeterministic way. The compartments are delimited by membranes, and the communication between them is essential.

A membrane system, also called a P system, consists of a hierarchy of membranes which do not intersect, with a distinguishable membrane (called skin) surrounding them all. The membranes produce a demarcation between regions (compartments). Regions contain multisets of objects, evolution rules, and possibly other membranes. Only rules in a region delimited by a membrane act on the objects in that region. The multisets of objects from a region correspond to the "chemicals swimming in the solution in the cell compartment", while the rules correspond to the "chemical reactions possible in the same compartment". The membrane systems are parallel and distributed. A detailed description of the P systems can be found in Ref. [1].

In this paper we study the assembly of asynchronous membrane systems from simpler ones. The way of composing P systems is inspired by the developments in the field of asynchronous processor design. We consider the most general class of asynchronous systems, namely the delay-insensitive systems. We outline the assembly of delay-insensitive complex P systems from simple functional components. P systems can be easily assembled into delay-insensitive systems if we allow antiport rules and promoters. Other ways can be considered, resulting in much more complicated communication and control-transfer rules, or more inefficient systems.

1 Asynchronous processors

Classical synchronous processors assume a clock signal that generates a rhythmic heartbeat for its components, synchronizing their functioning phases. While this makes the processor easy to design, there is a penalty to be paid in terms of power consumption, and functioning speed. A superior power consumption is caused by the clock distribution mechanism, and also by the fact that the functioning processor is never at an energetic equilibrium. That is, even if there are no input signals, the processor still performs idle instructions (e.g., global clock signals). While the idle instructions require less power than regular instructions, over a longer period of time a large portion of the power can be lost through these idle loops.

In contrast, asynchronous processors, also known as clockless processors, do not have a clock signal to synchronize their components. In asynchronous processors, when an input appears, the computational resources necessary for its processing are activated component by component until the input is completely processed. If the asynchronous processor has no input, the processor is in an energetic equilibrium and does not consume power. Thus, a lot of energy is saved, making asynchronous processors par-

* Supported by the research grants CEEX 47/2005 and CNCSIS 632/2006.

** To whom correspondence should be addressed. E-mail: gabriel@info.uaic.ro

ticularly useful for low-power embedded applications, e.g. wireless sensor networks.

Another advantage of asynchronous processors is the fact that each component can operate at maximal speed, because there is no need to synchronize on a clock tick with possibly slower components. This may prove useful in fast real-time systems, e.g. automotive systems.

The disadvantage of asynchronous processors is the much more complicated design technique. Examples of fully asynchronous processors are described and designed by Sutherland[2].

2 Asynchronous P systems

In this section we define asynchronous membrane systems. Before we define the compositional asynchronous P systems, we briefly review some notions and concepts related to asynchronous systems.

Definition 1. Delay-insensitive systems are systems in which there are no assumptions about the processing time of the components (modules), or about the transmission time along the connections.

Delay-insensitive systems are true asynchronous systems. Since in practical electronic designs they incur a severe efficiency penalty, quasi-delay-insensitive systems are usually preferred. However, given the abstract nature of P systems, we consider the delay-insensitive compositional P systems, because they have the highest degree of asynchrony and, as a consequence, also the highest modularity. This means that some components may take a longer or much shorter time to process an input (even the same input), and still the end result is the same. In other words, the system reacts consistently to the input. The power of delay-insensitive systems lies in the fact that once a component performs a certain operation correctly, it can be plugged in the system if the input and output are operated and interpreted in the required way. This allows the usage of a black-box development model in order to obtain a clean, modular design of the system. We consider the delay-insensitive P systems as an important class of compositional asynchronous P systems.

Definition 2. A delay-insensitive P system (DIPs) is a delay-insensitive system for which all the components are membrane systems.

Remark 1. The connections between components in a DIPs can be implemented by using antiport rules when we work with P systems, or synapses between cells when we work with tissue P systems.

Definition 3. Two P systems Π and Π' are result-equivalent if the output of both P systems is identical.

Proposition 1. The delay-insensitive P system produces the same result for the same input whenever we replace any component membranes with a result equivalent one.

Proof. Since delay-insensitive P systems are a class of delay-insensitive systems with membrane systems as components, by Definition 1 we can replace any component membranes with a result-equivalent one and the resulting DIPs is result-equivalent with the initial one.

Similarly to the electronic implementation of delay-insensitive systems, a handshake mechanism is needed to achieve the cooperation between components.

Definition 4. A handshake mechanism is an exchange of signals between two devices when communications begin, in order to ensure synchronization which precedes the information transfer.

Simply, whenever an input appears for a certain component, it does its part of processing, handing the output further to the next component(s).

Asynchronous systems manifest a high degree of robustness, because new components can be tested separately and added to the complex system by ensuring only the correct input and output interpretation. However the delay-insensitive systems incur a significant overhead, because the system must ensure that no input enters a component until the possibly pending processing in the component is finished. This can be resolved by adding a handshake mechanism between successive components to ensure that each component obeys the sequence of states presented in Fig. 1.

In order to propagate execution and input through the entire computational pathway, we use an antiport handshake, and signal-promoters similar to those presented in [3]. In P systems, an antiport rule has the form $(a, \text{out}; b, \text{in})$, which means that

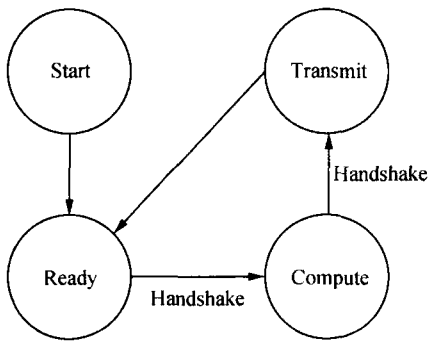


Fig. 1. FSM for a component.

an object a from the current region and an object b from the outer region can cross the delimiting membrane at the same time[4]. A promoter is a special object in a region which by its presence makes it possible to use a rule from its associated promoted rules set[5]; e.g. $a \to b \mid_p$ means that the rule $a \to b$ can be used in the presence of a promoter p.

By using antiport rules we can realize the synchronisation between successive components through the exchange of signal promoters which further facilitate the data and control transfer. In the following we present an algorithm implementing the data and control transfer between successive components.

3 Composition using antiport handshakes

Consider an n-tuple of functional P systems, each computing the values of a function. A complex system is constructed by aggregating these P systems by introducing each membrane inside the previous one. Membrane i computes the result of a function $f_i: M_i \to f_i(M_i)$, where M_i represents a multiset, and $i = \overline{0, n-1}$. A compositional functional P system as described before is valid if $\mathrm{Img}(f_i) \subseteq \mathrm{Dom}(f_{i+1})$, for $i = \overline{0, n-2}$. The complex P system in Fig. 2 computes the result of $f_{n-1} \circ \cdots \circ f_0$ applied to the input represented by the multiset in membrane 0 (the input membrane), and the results are obtained in membrane $n-1$ (the output membrane).

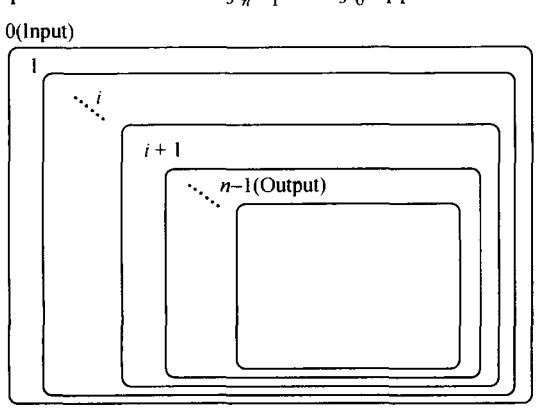


Fig. 2. A compositional P system.

Let us consider the membrane i, where $i = \overline{0, n-1}$. It must follow the evolution steps given by the following algorithm:

Algorithm 1. The delay-insensitive (DI) algorithm.

1. An antiport handshake ensures that membrane $(i-1)$ has finished the transmission;

2. membrane i performs its computation phase;

3. an antiport handshake ensures that membrane $(i+1)$ is ready to receive, and membrane i is ready to transmit;

4. membrane i transmits its output as input to membrane $(i+1)$;

5. membrane i goes back to ready state.

Remark 2. If $i = 0$, then step 1 is missing; if $i = n-1$, then steps 3 and 4 are missing because i is the last membrane on the path (the output membrane).

3.1 Data and control transfers between membranes

We describe the data and control transfers between membranes by considering the following special cases.

3.1.1 Intermediate membranes

When $i = \overline{1, n-2}$ and we do not work with the first or the last membrane, the data and control transfer proceeds as in Fig. 3. The order of the processes in the membranes is revealed by reading them from top to bottom. Following the DI algorithm, after the membrane i gets its input from the previous one, it performs its partial computation and is ready to communicate with membrane $i+1$, as soon as this membrane is ready for communication. After the antiport handshake between i and $i+1$ is completed, i starts transmitting its output as input to $i+1$, and after completion it enters the ready state. Control is now transferred to membrane $i+1$, and here the computation is performed now. Then a handshaking with the next membrane is attempted.

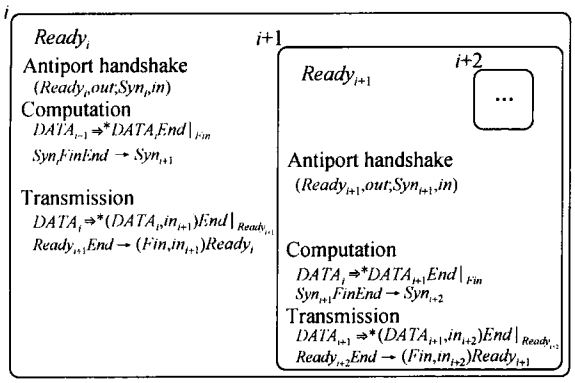


Fig. 3. From membrane i to membrane $(i+1)$.

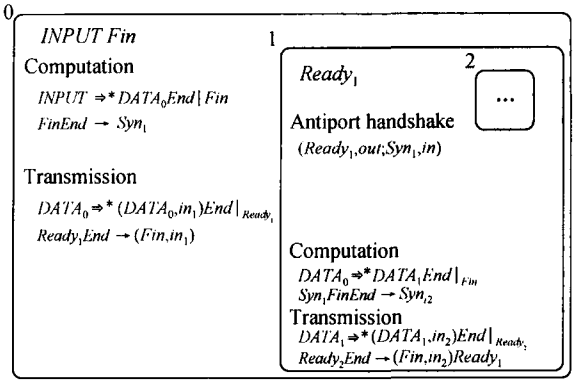


Fig. 4. From the first membrane to the second.

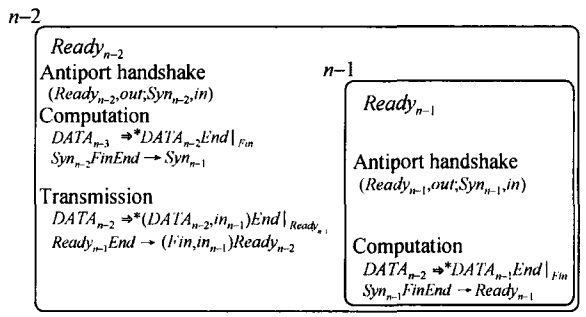


Fig. 5. End of the pathway.

Note that there are two types of rules and objects, those related to communication (antiport handshake, data transfer, and end transmission), and rules which perform computation inside a membrane.

Here is the communication pattern, presented separately:

1. $Syn_i FinEnd \rightarrow Syn_{i+1}$, which fires only after the computation in membrane i is finished; it creates an object Syn_{i+1} which serves as a connection request to membrane $i+1$;

2. antiport handshake: $(Ready_{i+1}, out; Syn_{i+1}, in)$ in the inner membrane;

3. data transfer, represented by the (possibly multi-step) evolution of the multiset $DATA_i$ into the multiset $DATA_i$ inside membrane $i+1$ and an object End in the current membrane:

$$DATA_i \Rightarrow^* (DATA_i, in_{i+1})End \mid_{Ready_{i+1}}$$

By this notation we understand that each rule which facilitates the transition is promoted by the object $Ready_{i+1}$. We note that an object End must be generated only after all data transfers are done, marking the end of this phase.

4. end of transmission is realized by $Ready_{i+1}$ $End \rightarrow (Fin, in_{i+1})Ready_i$, a rule in the outer membrane which removes the promoter $Ready_{i+1}$ and the object End (which signals the end of data transmission), and sends the signal-promoter Fin to the inner membrane as termination of the data transfer step. This rule also resets membrane i to the ready state.

3.1.2 First and last membranes

Data and control transfer for the first and the last membrane is presented in Figs. 4 and 5, respectively.

The input membrane 0 has no antiport handshake, since the input is stored directly into it, leaving only the computation and transmission phases. The rule $INPUT \Rightarrow^* DATA_0 \mid_{Fin}$ means that the input multiset evolves, in 0 or more steps, to an intermediate result of the computation. This result is then sent to membrane 1 via the transmission rules. We can note that a promoter Fin should be present in the membrane at the start of its functioning, being supplied at the end of the input insertion.

Remark 3. The last membrane $n-1$ lacks a transmission phase (see Fig. 5).

If desired, a read-output phase could be added at the end of the computation, which means that we need to process the results from the output membrane before producing another output.

3.2 System composition guide

While the former subsection deals with the design of the asynchronous protocol, handshake, data and control transfer, this one provides details on how to use such a compositional system. The user of the asynchronous protocol can design such an asynchronous chain of computational P systems (computational pathway) by fulfilling the following require-

ments:

1. The user must provide the goal-specific rule sets for computation and transmission. These rules must not use the control objects *Syn*, *Ready*, *End*, *Fin* with two exceptions which are described at 3. The rules are also confined to use objects from, and create objects in the embedding membrane.

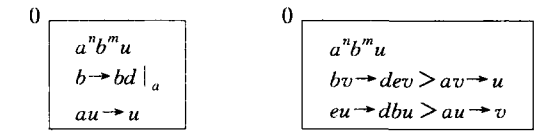
2. The user must ensure that the function computed by an asynchronous P system receives the proper input from the previous component, and produces the proper output for the next component.

3. The user must ensure that the computation and transmission phases produce an *End* object upon termination. Without the first requirement the system is not asynchronous, since the computation could be limited to a single step (or a fixed number of steps), while without the second one, the transmission to the next component could be limited to a single step, so the delay-insensitive system becomes only a speed-insensitive system.

We have used the above asynchronous protocol to compose some previously studied P arithmetic systems to create composite P systems able to compute arithmetic expressions. Please note that using this method of composition, there are no restrictions on how a certain component performs its computation, as long as it produces the correct result and obeys the above rules for asynchronous coupling with the rest of the system. The component P systems can be synchronous or asynchronous, they can be slower or faster, in the sense of steps required to compute the result; they can even belong to different P system families. The whole system remains asynchronous in the sense that each component waits for its neighbour to finish the computation before passing another input to it.

3.3 Computing arithmetic expressions

As a case study we developed the P systems for computing the arithmetical expression $n \cdot (m + p)$. We have two different implementations based on two multiplier P systems (Fig. 6), with or without promoters, as were presented in [6]. The two variants for multiplication have different time complexity, the one with promoters being faster.



(a) Multiplier with promoters (b) Multiplier w/o promoters

Fig. 6. Multiplier P systems.

We build the compositional asynchronous P system for computing $n \cdot (m + p)$ starting from its reverse polish notation $\cdot n + mp$. To compute this expression we must compose an adder P system with a multiplier P system. The results (the objects y represent the value of sum $m + p$) from the adder are sent to the multiplier as an input. After the computation phase, the transmission phase is started. We ensure that the computation in the inner membrane does not start until the transmission phase is finished. If the input is received in the multiplier, in the computation phase, the product $n \cdot y$ will be computed and the value of the expression $n \cdot (m + p)$ is obtained.

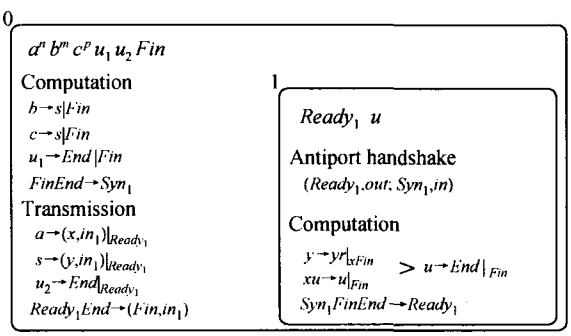


Fig. 7. Using multiplier with promoters.

In Fig. 7 the product $n \cdot y$ is computed with a multiplier P system with promoters of complexity $O(n)$. In Fig. 8 this product is computed with a multiplier P system without promoters of complexity $O(n \cdot m)$.

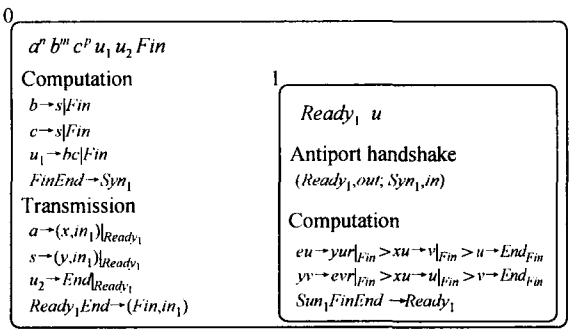


Fig. 8. Using multiplier without promoters.

The results of both P systems are the same. The difference between the two P systems is given by the computation rules of membrane 1. We can observe that if a component is replaced by another one which implements the same operation with a different complexity, the compositional P system produces the same result, because it is a delay-insensitive system.

Theorem 1. The compositional P systems constructed by using the delayinsensitive algorithm are delay-insensitive systems.

Proof. Consider a compositional P system

$$\Pi = (V, \mu, w_0, w_1, \cdots, w_i, \cdots, w_{n-1}, (R_0, \rho_0), (R_1, \rho_1), \cdots, (R_i, \rho_i), \cdots, (R_{n-1}, \rho_{n-1}), n-1)$$

$$\mu = [_0[_1 \cdots [_i \cdots [_{n-1}]_{n-1} \cdots]_i \cdots]_1]_0$$

$$R_i = R_i^h \cup R_i^c \cup R_i^t$$

If R_i is the set of evolution rules in membrane i, by R_i^h we denote the rules which implement handshake phase, by R_i^c we denote the rules which implement the transmission phase, and by R_i^t we denote the rules which implement the computation phase.

In Π we replace the membrane i with another membrane i' with a different, but result-equivalent computation and transmission phases, and we show that the resulting system Π' is result-equivalent to Π. In Π', we have $R_{i'} = R_i^h \cup R_{i'}^c \cup R_{i'}^t$ where $R_{i'}^c \cup R_{i'}^t$ is result-equivalent to $R_i^c \cup R_i^t$.

According to Definition 3, the systems Π and Π' are result-equivalent whenever the output of membrane i is the same as the output of membrane i', membrane $i+1$ has the same input in Π and Π'. Since both computational pathways starting from $i+1$ and up to $i-1$ in Π and Π' are identical, the same result is obtained in membrane $n-1$ in both Π and Π'. This holds for all i, and according to Proposition 1 we conclude that Π is a delay-insensitive P system.

4 Conclusion

In this paper we describe how we can build complex membrane systems from simpler ones. In this way we define compositional P systems, and in this way we overpass a weakness of the P systems when they are compared to compositional systems as brane calculus[7]. The rather algorithmic procedure of composing the simpler P systems seems particularly valuable in the case of asynchronous membrane systems, since the resulting membrane system remains asynchronous. The composition method is based on a handshake mechanism implemented by using antiport rules and promoters.

A great number of variants of synchronous P systems were explored, starting with transitional P systems, P systems with active membranes, with membrane polarization, with symport/antiport rules, with promoters and inhibitors and many others. Asynchronous P systems were also explored, but they do not refer to compositional asynchronous P systems as they are defined here. By using antiport rules as a handshake mechanism, we define the compositional asynchronous P systems in a simple way. This is simpler than in electronics, where for asynchronous systems there are two or four phase asynchronous interface protocols, whereas in our approach, antiport rules allow a single phase handshake protocol.

The delay-insensitive systems are similar to the systems described in [3, 8] under the names "time-free", "clock-free" or "time-independent". It will be useful to compare these systems, and see what are the common results, as well as the differences between these systems.

References

1 Păun Gh. Membrane Computing. An Introduction. Berlin: Springer-Verlag, 2002
2 Sutherland IE. Micropipelines. Communications ACM, 1989, 32: 720—738
3 Cavaliere M. Towards asynchronous P systems. In: Pre-proceedings Workshop on Membrane Computing WMC5, Milano, 2004, 161—173
4 Păun A and Păun Gh. The power of communication: P systems with symport/antiport. New Generation Computing, 2002, 20: 295—306
5 Bottoni P, Martin-Vide C, Păun Gh, et al. Membrane systems with promoters/inhibitors. Acta Informatica, 2002, 38: 695—720
6 Bonchiș C, Ciobanu G and Izbașa C. Encodings and arithmetic operations in membrane computing. In: Theory and Applications of Models of Computation Lecture Notes in Computer Science Vol. 3959, Berlin: Springer Verlag, 2006, 621—630
7 Cardelli L. Abstract machines of systems biology. Transactions on Computational Systems Biology III, 2005, 145—168
8 Cavaliere M and Sburlan D. Time-independent P Systems. In: Membrane Computing WMC5, Lecture Notes in Computer Science Vol.3365, Berlin: Springer Verlag, 2005, 239—258