

Computing with Membranes

Peranandam, Prakash Mohan*
Computational Logic Group
TU- Dresden
Germany

Project Guide
Prof.Dr.rer.nat.habil.Horst Reichel
TU Dresden
Fakultat Informatik
Institut fur Theoretische Informatik
D - 01062 Dresden
Germany

18th April 2001

*pp460034@irz.inf.tu-dresden.de

Abstract

AIM: The aim of this project is to study “Computing with Membranes” a new paradigm of computation and to develop some typical applications. Using an experimental simulator, written in Prolog, some experiments should be performed.

This paper presents one of the typical applications of Computing with Membranes introduced in [1]. Using an experimental simulator, written in Prolog presented in [2].

In Membrane Computing, Using the membranes as a kind of filter for specific objects when transferring them into an inner compartment turned out to be a very powerful mechanism. This is done in combination with suitable rules to be applied within the membranes in the model of generalised P-systems otherwise called Membrane Computing.

P-systems are computing models, where objects can evolve in parallel within a hierarchical membrane structure. Recent results show that this model is a promising framework for solving NP-complete problems in polynomial time.

The evolution rules are applied in a maximally parallel manner, at each step, all the objects which can evolve should evolve. A computation device is obtained, that start from an initial configuration and let the system evolve , a computation halts when no further rule can be applied. The set of objects in a specified output membrane are the result of the computation.

The typical application that presented here is Implementation of IP layer in TCP/IP protocol and Alternating-Bit protocol and tested using the simulator written in prolog. Both the protocols are compared and checked for strong bisimulation as explained in [3].

In this paper the implementation means, just implementing the concept level of protocols. and we don't consider in deep like packet size, address bits, etc.

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction to Computing with Membranes | 5 |
| 1.1 | Computing with Membranes | 5 |
| 1.2 | Some General Prerequisites | 5 |
| 1.3 | Membrane Structure | 6 |
| 1.4 | Super Cell System | 8 |
| 1.5 | Transition Super-cell Systems | 9 |
| 2 | Simulation environment written in prolog | 13 |
| 2.1 | Basic Operations in a Super Cell | 13 |
| 2.2 | Rules of the Super Cell System in Prolog Representation . . . | 14 |
| 3 | Implementation of IP Layer and Alternating-bit protocol | 14 |
| 3.1 | IP Layer | 15 |
| 3.1.1 | The Rules used for implementing IP layer | 15 |
| 3.2 | Alternating-bit protocol | 17 |
| 3.2.1 | Rules used for Implementing Alternating-bit protocol . | 18 |
| 3.3 | Rules and State transfer of Alternating-bit protocol | 21 |
| 3.3.1 | Gender's States and Rules | 21 |
| 3.3.2 | Receiver's States and Rules | 22 |
| 4 | Bisimulation | 23 |
| 4.1 | Bisimulation of the Protocols | 25 |
| 5 | An Example of Alternating-Bit protocol | 25 |
| 6 | Conclusion | 27 |

List of Figures

| | | |
|---|--|----|
| 1 | Venn diagram representation | 7 |
| 2 | An example of transitions in a super-cell system | 12 |
| 3 | Venn diagram representation with objects | 13 |
| 4 | Flow graph of Alternating-Bit Protocol | 17 |
| 5 | Transition diagram of Protocols | 23 |
| 6 | Transition diagram | 24 |

1 Introduction to Computing with Membranes

1.1 Computing with Membranes

The paper [1], [6], can be considered to be a contribution to Natural Computing, a field of research which tries to imitate the nature in the way it "Computes". It starts from the observation that any non-trivial biological system is a hierarchical construct, composed of several "organs" which are well defined and delimited from the neighbouring organs, which evolve internally and also cooperate with the other organs in order to keep alive the system as a whole.

In very particular terms, in biology and chemistry one knows membranes which keep together certain chemicals and leave to pass other chemicals, in a selective manner, some times only in one direction. Starting from these observations, the notion of membrane structure is considered as a mathematical counter part of hierarchical architectures composed of membranes recurrently distinguished in a given main membrane.

Computing with Membranes, based on notion of a membrane structure. such a structure consists of several cell-like membranes, recurrently placed inside a unique "SKIN" membrane. A plane representation is a Venn diagram without intersected sets and with a unique superset. In the regions delimited by the membranes there are placed objects, the obtained construct is called a super-cell. These objects are assumed to evolve according to certain rules, each object can be transformed in other objects, can pass through a membrane. A priority relation between evolution rules can be considered. The evolution is done in parallel for all objects able to evolve. In this way, we obtain a computing device.

1.2 Some General Prerequisites

In this we will specify some elementary notions and notations as per the explanation by Gheorghe Paun in [1], which will be useful for understanding the subsequent sections.

N - the set of natural numbers.

Let U be an arbitrary set. A multiset (over U) is a mapping $M: U \rightarrow N$; $M(a)$, for $a \in U$, is the multiplicity of 'a' in the multiset M . The multiplicity of each object with respect to any mutiset is finite. Note For a usual set $M \subseteq U$ we have $M(a) = 1$ when $a \in M$ and $M(a) = 0$ otherwise. The

support of a multiset M is the set $supp(M) = \{a \in U \mid M(a) > 0\}$. A multiset M is empty when its support is empty.

Let $M_1, M_2 : U \rightarrow N$ be two multisets. We say that M_1 is included in M_2 iff $M_1(a) \leq M_2(a)$, for all $a \in U$. The union of M_1 and M_2 is the multiset $M_1 \cup M_2 : U \rightarrow N$ defined by $(M_1 \cup M_2)(a) = M_1(a) + M_2(a)$, for all $a \in U$. The difference $M_1 - M_2 : U \rightarrow N$ given by $(M_1 - M_2)(a) = M_1(a) - M_2(a)$, for all $a \in U$.

A multiset M with a finite support $\{a_1, M(a_1), a_2, M(a_2), \dots, a_n, M(a_n)\}$, can be also represented by a string: $a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)}$ and all permutations of this string precisely identify the objects in the support of M and their multiplicities.

An alphabet is a finite nonempty set of abstract symbols. Given an alphabet V , we denote by V^* the sets of all finite strings of elements in V , including the empty string, λ . The length of a string $x \in V^*$ is denoted by $|x|$. A set of strings (over an alphabet V) is called a language (over V).

1.3 Membrane Structure

Consider now the following relations on MS : for $x, y \in MS$ we write $x \sim y$ if and only if we can write the two strings in the form $x = [1 \dots [2 \dots]_2 [3 \dots]_3 \dots]_1$, $y = [1 \dots [3 \dots]_3 [2 \dots]_2 \dots]_1$. we also denote by \sim the reflexive and transitive closure of the relation \sim . This is clearly an equivalence relation. We denote by \bar{MS} the set of equivalence classes of MS with respect to this relation. The elements of \bar{MS} are called membrane structures.

We stress the fact that when speaking of a membrane structure we do not take in to account the order of the membrane structures which are used when a new membrane structure is constructed and that each membrane structure is bracketed by an external pair $[]$ of parentheses.

It is easy to see that the parentheses $[,]$ appearing in a membrane structure are correctly matching, in the usual sense. Conversely, any string of correctly matching pairs of parentheses $[,]$ with a matching pair at the ends, corresponds to a membrane structure. Therefore we can write

$$MS = [D],$$

Where D is the Dyck language over $\{[,]\}$, that is, the language generated by the context-free grammar with the productions

$$S \rightarrow SS, S \rightarrow [S], S \rightarrow \lambda.$$

Each matching pair of parentheses $[,]$ appearing in a membrane structure is called a membrane. The number of membranes in a membrane structure μ is called the *degree* of μ and denoted by $deg(\mu)$. The external membrane of a membrane structure μ is called the skin membrane of μ . A membrane which appears in $\mu \in \bar{M}s$ in the form $[]$ (no other membrane appears inside the two parentheses) is called an elementary membrane.

The depth of a membrane structure μ , denoted by $dep(\mu)$, is defined recurrently as follows:

- if $\mu = []$, then $dep(\mu) = 1$;
- if $x = [\mu_1, \dots, \mu_n]$, for some $\mu_1, \dots, \mu_n \in MS$, then $dep(\mu) = \max\{dep(\mu_i) | 1 \leq i \leq n\} + 1$.

A membrane structure can be represented in a natural way as a Venn diagram. This makes clear the fact that the order of membrane structures of the same level in a larger membrane structure is irrelevant; What matters is the topological structure, the relationships between membranes.

The Venn representation of a membrane structure μ also makes clear the notion of a region in μ : any closed space delimited by membranes is called a region of μ . It is clear that a membrane structure of a degree 'n' contains n regions, one associated with each membrane.

Membrane Structure in Venn diagram representation is shown in figure 1

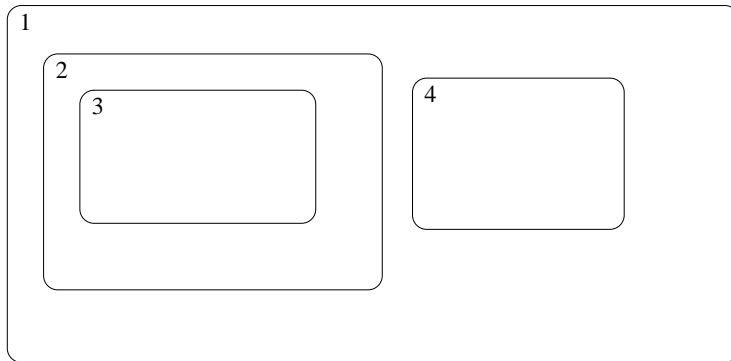


Figure 1: Venn diagram representation

1.4 Super Cell System

We now see how the definition of objects in a membrane structure. Let U be a denumerable set of objects. We call U the universe of our investigation and its elements are called objects.

Let μ be a membrane structure of degree n , $n \geq 1$, with the membranes labelled in a one-to-one manner, for instance, with the numbers from 1 to n . In this way, also the regions of μ are identified by the numbers from 1 to n . If a multiset $M_i : U \rightarrow N$ is associated with each region i of μ , $1 \leq i \leq n$, then we say that we have a super-cell.

Any multiset M_i mentioned above can be empty. In particular, all of them can be empty, that is, any membrane structure is a super-cell. On the other hand, each individual object can appear in several regions, in several copies in each of them.

Several notions defined for membranes structures are extended in the natural way to super-cells: degree, depth, region, etc.

The Multiset corresponding to a region of a super-cell (in particular, it can be an elementary membrane) is called the contents of it. The total multiplicities of the elements in an elementary membrane m (the sum of their multiplicities) is called the size of m and is denoted by $\text{size}(m)$.

If a membrane m' is placed in a membrane m such that they contribute to delimiting the same region, then all objects placed in the region associated with m are said to be adjacent to membrane m' . So, they are immediately "outside" membrane m' and "inside" membrane m .

The support of a super-cell π , denoted by $\text{supp}(\pi)$ is the set of all objects appearing in π at least once.

So Super-cell System is nothing else than a membrane structure with certain objects placed in regions delimited by the membranes. Because several copies of the same object can appear in the same region, multisets are considered. If the objects of a super cell are able to evolve, then computing device is obtained, and its called super-cell system.

Thus a super-cell system is a membrane structure with objects in its membranes, with specified rules for objects, and with given input-output prescriptions. Any object, alone or together with one more object evolves, can be transformed in other objects, can pass through one membrane, and can dissolve the membrane in which it is placed. (In this paper, we don't deal with dissolving of membranes for details refer [1]). All objects evolve at the same time, in parallel, in turn, all membranes are active in parallel. The

evolution rules are hierarchized by a priority relation, given in the form of a partial order relation; always, the rule with the highest priority among the applicable rules is actually applied.

An attractive feature of super-cell systems is their intrinsic parallelism. All objects-having access to a rule should use that rule.

1.5 Transition Super-cell Systems

We now introduce the main subject of the investigation, a computing mechanism essentially designed as a distributed parallel machinery, having as the underlying structure a super-cell. The basic additional feature is the possibility of objects to evolve, according to certain rules.

A transition super-cell system of degree n , $n \geq 1$, is a construct

$$\Pi = (V, \mu, M_1, \dots, M_n, (R_1, \rho_1), \dots, (R_n, \rho_n), i_0),$$

where

- V is an alphabet; its elements are called objects;
- μ is a membrane structure of degree n , with the membranes and the regions labelled in a one-to-one manner with elements in a given set Λ ; in this section we always use the labels $1, 2, \dots, n$;
- M_i , $1 \leq i \leq n$, are multisets over V associated with the regions $1, 2, \dots, n$ of μ ;
- R_i , $1 \leq i \leq n$, are finite sets of evolution rules over V associated with the regions $1, 2, \dots, n$ of μ ; ρ_i is a partial order relation over R_i , $1 \leq i \leq n$, specifying a priority relation among rules of R_i . An evolution rule is a pair (u, v) , which we will usually write in the form $u \rightarrow v$, where u is a string over V and $v = v'$ or $v = v'\delta$, where v' is a string over

$$(V \times \{here, out\}) \cup (V \times \{in_j | 1 \leq j \leq n\}),$$

and δ is a special symbol not in V . The length of u is called the radius of the rule $u \rightarrow v$.

- i_0 is a number between 1 and n which specifies the output membrane of Π .

Any of the multisets M_1, \dots, M_n can be empty and the same is valid for the sets R_1, \dots, R_n and their associated priority relations ρ_i .

The components μ and M_1, \dots, M_n of a super-cell system define a super-cell. Graphically, we will represent a super-cell system by representing its underlying super-cell, and also adding the rules to each region, together with the corresponding priority relation. In this way, we can have a complete picture of a super-cell system, much easier to understand than a symbolic description.

The components $\mu, M_1, \dots, M_n, (R_1, \rho_n)$ constitute the initial configuration of Π . In general, any sequence $\mu', M'_{i_1}, \dots, M'_{i_k}, (R_{i_1}, \rho_{i_1}), \dots, (R_{i_k}, \rho_{i_k})$, with μ' a membrane structure obtained by removing from μ all membranes different from i_1, \dots, i_k (the skin membrane is not removed), with M'_j multisets over V , $1 \leq j \leq k$, and $\{i_1, \dots, i_k\} \subseteq \{1, 2, \dots, n\}$, is called a configuration of Π .

A more compact and easy to read writing of a configuration, avoiding the use of subscripts for multisets and sets above is that where the objects of the multisets are written (using multisets or in the form of a string) directly in the region to which they belong, and, similarly, the rules are written in the region where they can act. This is in a good correspondence with the graphical representation of a transition super-cell system and we will use it especially for configurations where many components are empty.

For two configurations

$$\begin{aligned} C_1 &= \mu', M'_{i_1}, \dots, M'_{i_k}, (R_{i_1}, \rho_{i_1}), \dots, (R_{i_k}, \rho_{i_k}), \\ C_2 &= \mu'', M''_{j_1}, \dots, M''_{j_l}, (R_{j_1}, \rho_{j_1}), \dots, (R_{j_l}, \rho_{j_l}) \end{aligned}$$

of Π we write $C_1 \longrightarrow C_2$ and say that we have a transition from C_1 to C_2 , if we can pass from C_1 to C_2 by using the evolution rules appearing in R_{i_1}, \dots, R_{i_k} in the following manner.

Consider a rule $u \rightarrow v$ in a set R_{i_t} . We look to the region of $/mu'$ associated with the membrane i_t . If the objects mentioned by u , with the multiplicities specified by u , appear in M'_{i_t} , then these objects can evolve according to the rule $u \rightarrow v$. The rule can be used only if no rule of a higher priority exists in R_{i_t} and can use the objects in u . More precisely, we start to examine the rules in the decreasing order of their priority and assign objects

to them. A rule can be used only when there are copies of the objects whose evolution it describes and which were not "consumed" by rules of a higher priority. Therefore, all objects to which a rule can be applied must be the subject of a rule application. All objects in u are "consumed" by using the rule $u \rightarrow v$, that is, the multiset identified by u is subtracted from M'_{i_t} .

The result of using the rule is determined by v . If an object appears in v in a pair (a, here), then it will remain in the same region i_t . (Often, when specifying rules, pairs (a, here) are simply written 'a', the indication "here" is omitted.) If an object appears in v in a pair (a, out), Then 'a' will exit the membrane i_t and will become an element of the region immediately outside it (thus it will be adjacent to the membrane i_t from which it was expelled). In this way it is possible that an object leaves the super-cell itself: if it goes outside the skin of the super-cell, then it never comes back. If an object appears in a pair (a, in_q), then 'a' will be added to the multiset M'_q , providing that 'a' is adjacent to the membrane q . If (a, in_q) appears in v and the membrane q is not one of the membranes delimiting "from below" the region i_t , then the application of the rule is not allowed.

If the symbol \S appears in v , then the membrane i_t is removed (we say dissolved) and at the same time the set of rules R_{i_t} (and its associated priority relation) is removed. We do not allow the dissolving of the skin, because this means that the super-cell is lost, we do no longer have a correct configuration of the system.

All these operations are done in parallel, for all possible applicable rules $u \rightarrow v$, for all occurrences of multisets u in the region associated with the rules, for all region at the same time.

In this paper we don't deal with dissolving of membranes. For more details refer paper [1]. Now, We will see an example explained by Gheorghe Paun to clarify the definition of a transition in a super-cell system.

Consider the super-cell system of degree 4:

$$\Pi = (V, \mu, M_1, \dots, M_4, (R_1, \rho_1), \dots, (R_4, \rho_4), 4),$$

$$V = \{a, b, c, d\},$$

$$\mu = [{}_1[{}_2[{}_3]_3]_2[{}_4]_4]_1,$$

$$M_1 = \{aac\},$$

$$M_2 = \{a\},$$

$$M_3 = \{cd\},$$

$$M_4 = \emptyset,$$

$$R_1 = \{r_1 : c \rightarrow (c, in_4), r_2 : c \rightarrow (b, in_4), r_3 : a \rightarrow (a, in_2)b, dd \rightarrow (a, in_4)\},$$

$$\rho_1 = \{r_1 > r_3, r_2 > r_3\},$$

$$R_2 = \{a \rightarrow (a, in_3), ac \rightarrow \S\},$$

$$\rho_2 = \emptyset,$$

$$R_3 = \{a \rightarrow \S\},$$

$$\rho_3 = \emptyset,$$

$$R_4 = \{c \rightarrow (d, out), b \rightarrow b\},$$

$$\rho_4 = \emptyset.$$

The example of transition in a super-cell system figure 2.

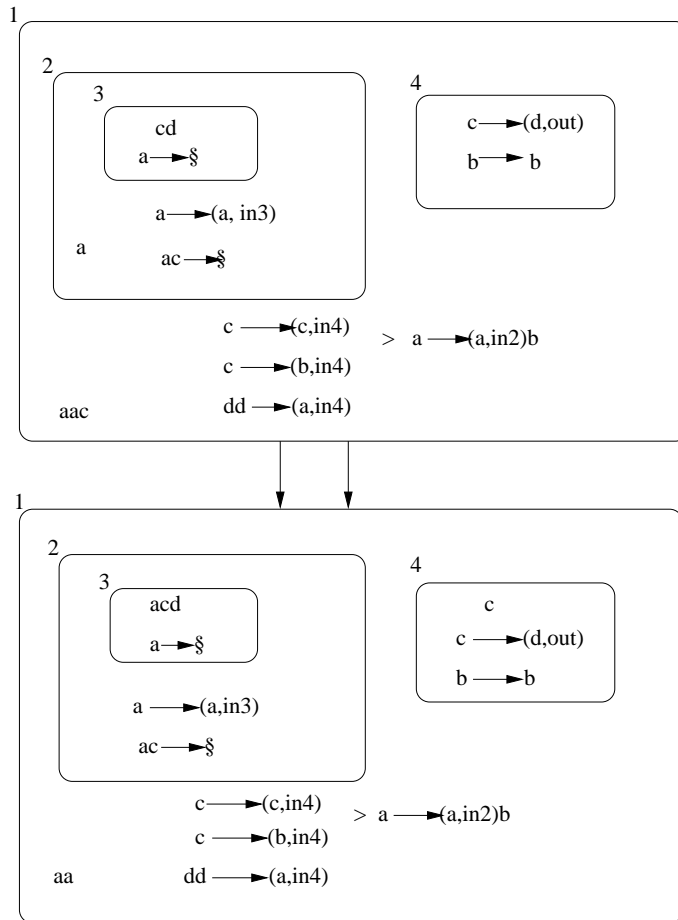


Figure 2: An example of transitions in a super-cell system

2 Simulation environment written in prolog

In this paper [2], Mihaela Malita presented the simulation environment written in prolog. Let us see how the membrane structure and basic operations of membrane computing was handled in prolog by M.Malita.

The Membrane structure in Prolog:

A Membrane is an labelled multiset. Example of a membrane structure: In Paun's first example $Ms = [a,a,c,[a,[c,d]],[]]$,

The membranes are labelled with numbers, so we could refer them in the super cell. so, the representation that we work with is:

$Ms = [1,a(2),c(1),[2,a(1),[3,c(1),d(1)],[4]]]$

Membrane Structure is shown in figure 3

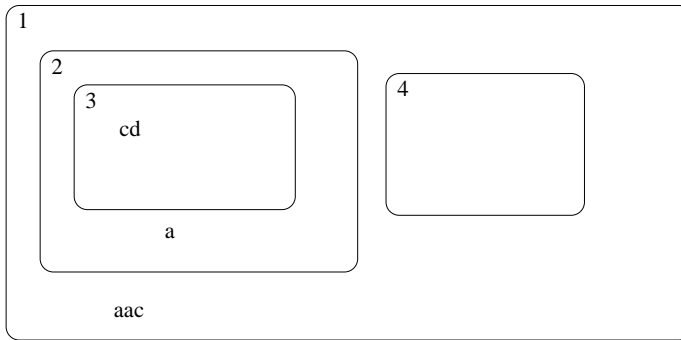


Figure 3: Venn diagram representation with objects

2.1 Basic Operations in a Super Cell

- $\text{union}(M1,M2,R)$ - R is the union of membranes M1 and M2. It follows the same pattern as the union for multisets.
- $\text{difference}(M1,M2,D)$ - D is the difference of membranes, If the multisets are not included one in another the difference is not possible(fails)
- $\text{membrane}(K,Ms,Mk)$ - Mk will be the content of membrane K from the membrane structure Ms
- $\text{transf}(K,M,Ms,R)$ - This predicate transforms the membrane structure Ms by replacing the old membrane K with a new membrane M resulting the membrane structure R.

2.2 Rules of the Super Cell System in Prolog Representation

Rule is represented in the for

- `rule(1,1,[c(1)],[in(4),a(1)])`.

The first 1 is the label of the membrane and the second 1 is the label of the rule. This rule means that if membrane 1 contains one object 'c' then we remove it from membrane 1 and we put an object 'a' in membrane 4.

Its also possible to have a rule for the same membrane:

In our representation

- `rule(4,1,[b(1)],[a(1)])`.

This means in membrane 4 we could change the object 'b' with object 'a'. Actually all the b's are transformed to a's. Another principle is: If a rule works for a membrane we apply the rule until it works no more.

3 Implementation of IP Layer and Alternating-bit protocol

A communication protocol is a discipline for transmission of messages from source to a destination. Typically, the adverse conditions pertain to the transmission medium; for example, the sender cannot assume that a transmission was successful until an acknowledgement is obtained from the receiver. Refer [4] for Network protocols.

We have considered a tree structure for representing the system and the network. If for example we consider a network where 4 terminals are connected, let us name the network as 1, and the terminals as 12,13,14,15.

so its represented as,

Membrane = `[1,[12],[13],[14],[15]]`

All the rule is represented in prolog format, i.e., the capital letters are variables and small letters are objects.

3.1 IP Layer

The Internetwork layer provides communication between computers. Part of communicating messages between computers is a routing function that ensures that message messages will be correctly delivered to their destination. The Internet Protocol(IP) provides this routing function.

IP provides two service primitives at the interface to the next-higher layer. The Send primitive of IP is used to request transmission of a data unit. The Deliver primitive is used by IP to notify a user of the arrival of a data unit.

IP is the layer that hides the underlying physical network from the upper-layer protocols. It is an unreliable , best-effort, and connection less packet delivery protocol. The best effort means that the packets sent by IP may be lost, or duplicated, but IP will not handle these situations. It is up to the higher-layer protocols to deal with these situations.

In this we have modelled the IP layer which send a packet from the source system to destination system. And then the destination system will send an acknowledgement for the packet it received, to the source system. Also packet lost is handled (part of TCP layers work), i.e. whenever a packet is not reached the destination (Packet lost), source will resend the same packet after certain time delay.

we have used some specific objects like

- 'send' - is used as send primitive object.
- 'ack' - is used as the acknowledgement object.
- 'tcp' - is used to represent that a packet is received from TCP layer.
- 'da(X)' - is used as a destination address object where X is destination address, which is a terminal on the network.
- 'sa(X)' - is used as a source address object where X is source address, which is a terminal on the network.
- 'p(X)' - is used as a packet representation, where X is a list of datas, that to sent from source to destination terminal.

3.1.1 The Rules used for implementing IP layer

- **Rule 1:** $\text{rule}(S,1,[\text{tcp},p([\text{h},\text{e},\text{l},\text{l},\text{o}]),\text{da}(D),\text{sa}(S)],[\text{send},p([\text{h},\text{e},\text{l},\text{l},\text{o}]),\text{da}(D),\text{sa}(S)])$.

This rule can be explained as, when ever there are objects 'tcp', packet 'p([X])', destination address 'da(D)', senders address 'sa(S)', in membrane S, then it will rewrite the object 'tcp' to 'send'.

Example: If the membrane structure is ,

Membrane = [1,[12],[13],[14],[15,tcp,p([h,e,l,l,o]),da(12),sa(15)]]

after applying the rule the resulting membrane structure will be,

Membrane = [1,[12],[13],[14],[15,send,p([h,e,l,l,o]),da(12),sa(15)]]

- **Rule 2:** rule(S,2,[send,X,da(D),sa(S)],[in(D),ack,X,da(D),sa(S)]).

This rule can be explained as when ever there are objects 'send', packet X, destination address 'da(D)', senders address 'sa(S)', in membrane S, it will pass the objects 'ack', packet 'X', destination address and senders address to the membrane D.

Example: If the membrane structure is ,

Membrane = [1,[12],[13],[14],[15,send,p([h,e,l,l,o]),da(12),sa(15)]]

after applying the rule the resulting membrane structure will be,

Membrane = [1,[12,ack,p([h,e,l,l,o]),da(12),sa(15)],[13],[14],[15]]

- **Rule 3:** rule(D,3,[ack,X,da(D),sa(S)],[in(S),ack,sa(D)]).

This rule can be explained as whenever there are objects 'ack', packet 'X', destination address and senders address, then the object ack and senders address will be passed to the membrane D. (Note: The senders address is the actual destination address, it interchanged as acknowledgement is passed), and it deletes the object 'ack' and destination address from the destination membrane.

Example: If the membrane structure is,

Membrane = [1,[12,ack,p([h,e,l,l,o]),da(12),sa(15)],[13],[14],[15]]

after applying the rule the resulting membrane structure will be,

Membrane = [1,[12,p([h,e,l,l,o]),sa(15)],[13],[14],[15,ack,sa(12)]]

- **Rule 4:** rule(S,4,[],[tcp,Pac,da(D),sa(S)]).

This rule can be explained as a time out for acknowledgement. Whenever there is no object 'ack'(acknowledgement) in the membrane S, for the packet it sent before 4 clock period, this rule will reassume the old

packet, inorder to resend the packet for which it didn't get acknowledgement.

Example: If the membrane structure is,

Membrane = [1,[12],[13],[14],[15]]

and we consider that, membrane 15 has sent a packet $p([h,e,l,l,o])$ to the destination address 12, but didn't get any acknowledgement for the past 4 clock period, then, after applying the rule the resulting membrane structure will be,

Membrane = [1,[12],[13],[14],[15,tcp,p([h,e,l,l,o]),da(12),sa(15)]]

3.2 Alternating-bit protocol

First, We assume that the packets may get lose or duplicate but not corrupt messages. The name of the protocol refers to the method used. Messages are sent tagged with bits 0 and 1 alternately, and these bits also constitute the acknowledgements.

The flow graph indicates the direction of information flow by arrows figure:4.

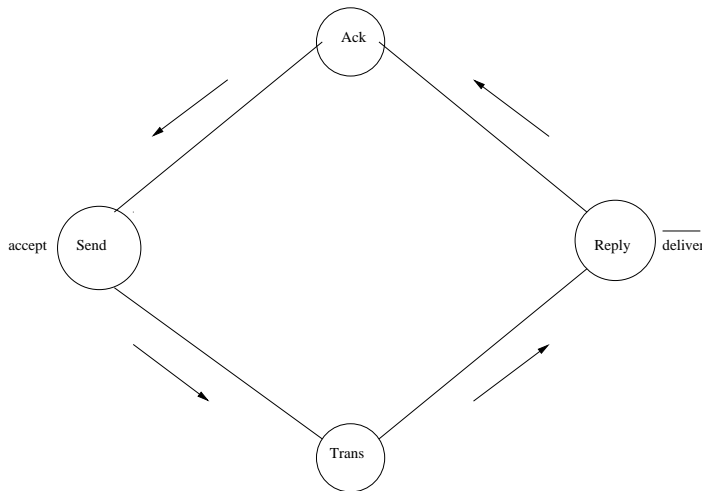


Figure 4: Flow graph of Alternating-Bit Protocol

The sender works as follows. After accepting a message, it sends it with bit b and sets a timer. There are three possibilities:

- It may get a 'time-out' from the timer, upon which it sends the message again with b;
- It may get an acknowledgement b from the sender, upon which it is ready to accept another message (which it will send with bit $b'=1-b$);
- It may get an acknowledgement b' (resulting from a superfluous retransmission of the previous message) which it ignores.

The replier works in a dual manner. after delivering a message it acknowledges it with bit b, and sets a timer. Then there are three possibilities:

- It may get a 'time-out' from the timer, upon which it acknowledges again with b;
- It may get a new message with bit b' from the sender upon which it is ready to deliver a new message (which it will acknowledge with bit b');
- It may get a superfluous transmission of the previous message with bit b, which it ignores.

In the implementation of this protocol, we use some special objects which is used to evolve according to the rules.

- send0, send1 - used as alternating bit with the packet when its evolved.
- ack0, ack1 - used to acknowledge the packets according to the order of reception.
- $p(X)$ - packet, where X is a list with unique serial number and the actual data in it.
- accept - used to accept next data, when acknowledgement is received.

3.2.1 Rules used for Implementing Alternating-bit protocol

- **Rule 1:** $\text{rule}(S, 1, [\text{accept}], [B, X, \text{da}(D), \text{sa}(S)])$.

This rule is to accept the new packet to be sent to the destination terminal. and its applied when ever there is a object 'accept' in the membrane S. The alternating bit '0' and '1' is combined with the object 'send' as 'send0' or 'send1'. This objects is accepted alternately.

Example: If the membrane structure is,

Membrane = [1,[12,[121]],[13,accept,[131]],[14],[15]]

after applying the rule the resulting membrane structure will be,

Membrane = [1,[12,[121]],[13,send0,p([1,h]),da(12),sa(13),[131]],[14],[15]]

- **Rule 2:** rule(S,2,[accept],[1]).

This rule is to end the accept, when the packet to be transformed is completed.

Example: If the membrane structure is,

Membrane = [1,[12,[121]],[13,accept,[131]],[14],[15]]

and all the packets are already sent, then this rule is applicable. The packets to be sent are stored before the process starts. And the number of clock period needed to complete the whole cycle is calculated, normally it is 4 clock period per packet.

after applying the rule the resulting membrane structure will be,

Membrane = [1,[12,[121]],[13,[131]],[14],[15]]

- **Rule 3:** rule(S,3,[B,X,da(D),sa(S)],[in(D),A,X,da(D),sa(S)]).

This rule is to transform the packet that was accepted by rule 1, to the destination address, with an object 'A', which is the acknowledgement to be sent back.

Example: If the membrane structure is,

Membrane = [1,[12,[121]],[13,send0,p([1,h]),da(12),sa(13),[131]],[14],[15]]

after applying the rule the resulting membrane structure will be,

Membrane = [1,[12,ack0,p([1,h]),da(12),sa(13),[121]],[13,[131]],[14],[15]]

- **Rule 4:** rule(D,4,[A,X,da(D),sa(S)],[in(S),A]).

This rule is to send the acknowledgement A back to the packet sender.

Example: If the membrane structure is,

Membrane = [1,[12,ack0,p([1,h]),da(12),sa(13),[121]],[13,[131]],[14],[15]]

after applying the rule the resulting membrane structure will be,

Membrane = [1,[12,p([1,h]),[121]],[13,ack0,[131]],[14],[15]]

- **Rule 5:** rule(S,5,[A],[accept]).

This rule is for, whenever acknowledgement A is received from the receiver, sender will rewrite acknowledgement to object accept, which will in turn accept a new packet.

Example: If the membrane structure is,

Membrane = [1,[12,[121,p([1,h])]], [13,ack0,[131]], [14],[15]]

after applying the rule the resulting membrane structure will be,

Membrane = [1,[12,[121,p([1,h])]], [13,accept,[131,ack0]], [14],[15]]

note that the object 'ack0' has been transferred to the cell 131 when the object is 'accept' is rewritten by the rule. this is to keep track on the acknowledgement what all it has received and in case if it gets the duplicate acknowledgement it identifies it and cancel the duplicate acknowledgement.

- **Rule 6:** rule(S,6,[],[B,X,da(D),sa(S)]).

This rule is to resend the packet if the time out happens. i.e, if a packet has been sent and if there is no acknowledgement received for the packet for some time limit then this rule is applied in order to resend the packet. And the packet and all other details to be sent again, would be stored temporarily, when it was sent for first time. Once the source gets the acknowledgement the temporary storage will be updated for the new packet sent.

Example: If the membrane structure is,

Membrane = [1,[12,[121]], [13,[131]], [14],[15]]

after applying the rule the resulting membrane structure will be,

Membrane = [1,[12,[121]], [13,send0,p([1,h]),da(12),sa(13),[131]], [14],[15]]

- **Rule 7:** rule(D,7,[p(X)],[in(121),p(X)]).

This rule is to deliver the packet, and its transformed to the internal cell or storage of the receiver.

Example: If the membrane structure is,

Membrane = [1,[12,p([1,h])[121]], [13,[131]], [14],[15]]

after applying the rule the resulting membrane structure will be,

Membrane = [1,[12,[121,p([1,h])]], [13,[131]], [14],[15]]

- **Rule 8:** $\text{rule}(D,8,[],[\text{in}(S),A])$.

This rule is to resend the acknowledgement, If the time out happens. i.e, if an acknowledgement has been sent and if there is new packet received by destination membrane for some time limit, then this rule is applied inorder to resend the acknowledgement. And the acknowledgement to be sent again would be stored temporarily, when it was sent for first time. Once the source gets the acknowledgement the temporary storage will be updated.

Example: If the membrane structure is,

Membrane = $[1,[12,[121]],[13,[131]],[14],[15]]$

after applying the rule the resulting membrane structure will be,

Membrane = $[1,[12,[121]],[13,\text{ack}0,[131]],[14],[15]]$

3.3 Rules and State transfer of Alternating-bit protocol

Let us discuss about Alternating-Bit protocol, that how the rules used in the implementation, resembles the state transitions of the real protocol.

3.3.1 Gender's States and Rules

As per alternating-Bit protocol 'sender' in figure 5, we will check the rules that are responsible for all the states its reaching.

- **State s0 to s1:** Rule 3 is the one which transfers the sender from state s0 to s1 by sending the packet to the destination terminal.
- **State s1 to s0:** Rule 6 is the one which transfers the sender from state s1 to s0 by performing time out which is 4 clock per packet.
- **State s1 to s2:** Rule 4 is the one which transfers the sender from state s1 to s2, by sending the acknowledgement, ack0 or ack1 from the destination terminal. The state transfer occurs only when the respective acknowledgement is received. This check is performed by storing the previous acknowledgements in the source terminal, and it should be alternating bit, i.e, ack0,ack1,ack0...etc should be the series of acknowledgement received by source terminal.

- **State s1 to s1:** Rule 4 is the one which holds the sender in the same state s1, when the duplicate acknowledgement is received. If some acknowledgement bit is received as same of the previous acknowledgement bit, then it simply rejects those acknowledgements and remains in the same state, still it gets the correct acknowledgement bit.
- **State s2 to s0:** Rule 1 is the one which transfers the sender state from s2 to s0, by accepting the new packet to be sent. In this transition there is a hidden state which is handled by rule 5, where the object 'ack0' or 'ack1' is rewritten as object 'accept'. Rule 2 is also used in the implementation just to stop the packet accept ion. This is to make the implementation work.

3.3.2 Receiver's States and Rules

As per the alternating-Bit protocol 'receiver' in figure 5 we will check the rules that are responsible for all the states its reaching.

- **State r0 to r1:** Rule 4 is the one which transfers the receiver from state r0 to r1, by by sending the acknowledgement, ack0 or ack1 to the source terminal.
- **State r1 to r0:** Rule 8 is the one which transfers the receiver from state r1 to r0 by performing time out which is 4 clock per acknowledgement.
- **State r1 to r2:** Rule 3 is the one which transfers the receiver from state r1 to r2, by sending the packets, from the source terminal. The state transfer occurs only when the new packet is received. This check is performed by storing the previous received packets in the destination terminal. (Note: All packets should have an unique identification serial number. In our representation, it is, $p([1,h]), p([2,a])...$ etc. There can be same data with different serial number.
- **State r1 to r1:** Rule 3 is the one which holds the receiver in the same state r1, when the duplicate packet is received. It simply rejects those duplicate packets and remains in the same state, still it gets the new packet, i.e packet with new serial number.
- **State r2 to r0:** Rule 7 is the one which transfers the receiver from state r2 to r0, by delivering the packet to its storage unit, and frees the receiver for new packet reception.

So we conclude that the Implementation of the Alternating-Bit protocol in membrane computing using rules is as specified.

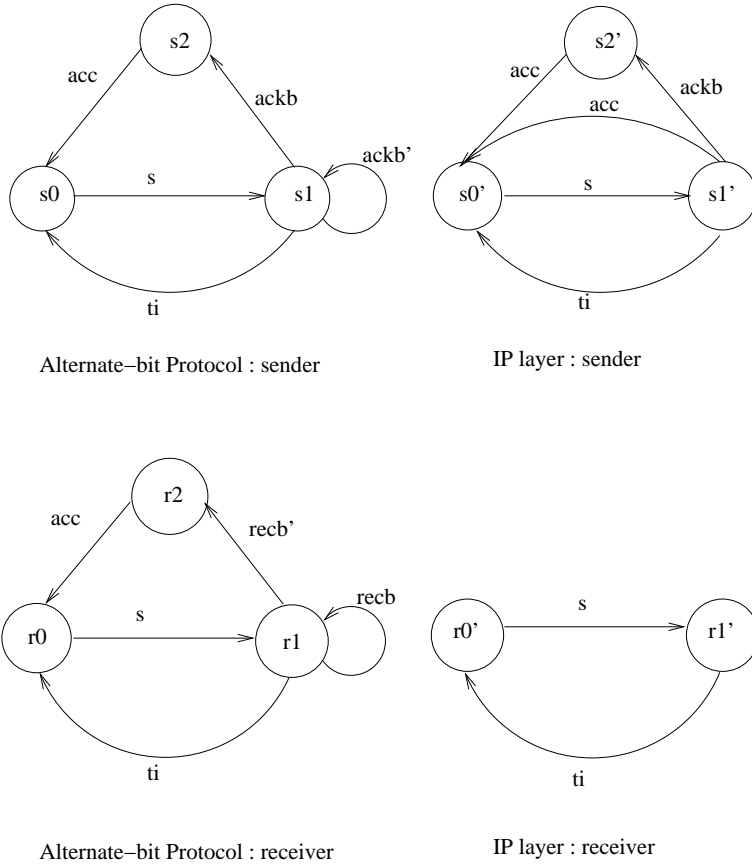


Figure 5: Transition diagram of Protocols

4 Bisimulation

In the book [5], Milner has explained about Bisimulation, It is based on a concept of mutual simulation, which is called Bisimulation.

we will see an example from the book, figure 6

Its argued that these should not be equivalent. To pinpoint the way they differ, The notion of simulation is introduced according to which p_0 can simulate q_0 , but not vice versa.

Definition 4.1 *Labelled transition system* A labelled transition system (LTS) over Act is a pair (Q, T) consisting of

- a set Q of states;
- a ternary relation $T \subseteq (Q \times \text{Act} \times Q)$, known as a transition relation.

If $(q, \alpha, q') \in T$ we write $q \xrightarrow{\alpha} q'$, and we call q the source and q' the target of the transition.

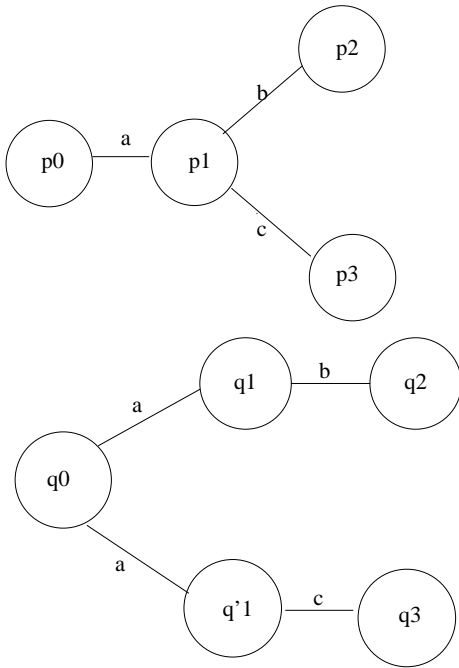


Figure 6: Transition diagram

Definition 4.2 *Strong Simulation* Let (Q, T) be an LTS (Labelled Transition System), and let S be a binary relation over Q . Then S is called a strong simulation over (Q, T) if, whenever $p S q$,

if $p \xrightarrow{\alpha} p'$ then there exists $q' \in Q$ such that $q \xrightarrow{\alpha} q'$ and $p' S q'$.

Definition 4.3 *Strong Bisimulation, Strong Equivalence* A binary relation S over Q is said to be a strong strong bisimulation over the LTS (Q, T) if both S and its converse are simulations. We say that p and q are strongly bisimilar or strongly equivalent, written $p \sim q$, if there exists a strong bisimulation S such that $p S q$.

4.1 Bisimulation of the Protocols

Looking at the transition diagrams for both protocol implemented, in figure 5, Its obvious that IP layer transitions is not even a strong simulation of Alternating-Bit protocol as per paper [3] by Prof.H.Reichel.

Let us see why its not even strong simulation. As per the definition we will check action by action and state by state.

Initially consider the strong simulation of the states s_0 and s_0' .

so the set $S = (s_0, s_0')$

There exists s_1 through the action s from s_0 . According to the definition there should be s_1' through the same action s . as its there let us include strong simulation states in the set S .

$S = (s_0, s_0'), (s_1, s_1')$

from the state s_1 there is 3 possibilities of next states. s_1, s_0, s_2 are the 3 possible next states. Obviously there is no possibility of going to s_1' from s_1' through a rule. so this cant be a strong simulation according to the definition.

5 An Example of Alternating-Bit protocol

Here is the output of the implementation, for 2 packets to be sent, which is 'h' and 'a'

How many generations ?=—: 10.

Membrane is[1,[12,[121]],13,accept,[131]],14],[15]]

Clock = 1

Membrane = [1,[12,[121]],13,accept,[131]],14],[15]]

RULE = [13,1][accept] → [send0,p([1,h]),da(12),sa(13)]

OLD = [1,[12,[121]],13,accept,[131]],14],[15]]

SUCCEDED ! New=[1,[12,[121]],13,send0,p([1,h]),da(12),sa(13),[131]],14],[15]]

Result = [1,[12,[121]],13,send0,p([1,h]),da(12),sa(13),[131]],14],[15]]

Clock = 2

Membrane = [1,[12,[121]],13,send0,p([1,h]),da(12),sa(13),[131]],14],[15]]

RULE = [13,3][send0,p([1,h]),da(12),sa(13)] → [in(12),ack0,p([1,h]),da(12),sa(13)]

OLD = [1,[12,[121]],13,send0,p([1,h]),da(12),sa(13),[131]],14],[15]]

SUCCEDED ! New=[1,[12,ack0,p([1,h]),da(12),sa(13),[121]],13,[131]],14],[15]]

Result = [1,[12,ack0,p([1,h]),da(12),sa(13),[121]],[13,[131]],[14],[15]]

Clock = 3

Membrane = [1,[12,ack0,p([1,h]),da(12),sa(13),[121]],[13,[131]],[14],[15]]

RULE = [12,4][ack0,p([1,h]),da(12),sa(13)] → [in(13),ack0]

OLD = [1,[12,ack0,p([1,h]),da(12),sa(13),[121]],[13,[131]],[14],[15]]

SUCCEDED ! New=[1,[12,p([1,h]),[121]],[13,ack0,[131]],[14],[15]]

RULE = [12,7][p([1,h])] → [in(121),p([1,h])]

OLD = [1,[12,p([1,h]),[121]],[13,ack0,[131]],[14],[15]]

SUCCEDED ! New=[1,[12,[121,p([1,h])]],[13,ack0,[131]],[14],[15]]

Result = [1,[12,[121,p([1,h])]],[13,ack0,[131]],[14],[15]]

Clock = 4

Membrane = [1,[12,[121,p([1,h])]],[13,ack0,[131]],[14],[15]]

RULE = [13,5][ack0] → [accept]

OLD = [1,[12,[121,p([1,h])]],[13,ack0,[131]],[14],[15]]

SUCCEDED ! New=[1,[12,[121,p([1,h])]],[13,accept,[131,ack0]],[14],[15]]

Result = [1,[12,[121,p([1,h])]],[13,accept,[131,ack0]],[14],[15]]

Clock = 5

Membrane = [1,[12,[121,p([1,h])]],[13,accept,[131,ack0]],[14],[15]]

RULE = [13,1][accept] → [send1,p([2,a]),da(12),sa(13)]

OLD = [1,[12,[121,p([1,h])]],[13,accept,[131,ack0]],[14],[15]]

SUCCEDED ! New=[1,[12,[121,p([1,h])]],[13,send1,p([2,a]),da(12),sa(13),[131,ack0]],[14],[15]]

Result = [1,[12,[121,p([1,h])]],[13,send1,p([2,a]),da(12),sa(13),[131,ack0]],[14],[15]]

Clock = 6

Membrane = [1,[12,[121,p([1,h])]],[13,send1,p([2,a]),da(12),sa(13),[131,ack0]],[14],[15]]

RULE = [13,3][send1,p([2,a]),da(12),sa(13)] → [in(12),ack1,p([2,a]),da(12),sa(13)]

OLD = [1,[12,[121,p([1,h])]],[13,send1,p([2,a]),da(12),sa(13),[131,ack0]],[14],[15]]

SUCCEDED ! New=[1,[12,ack1,p([2,a]),da(12),sa(13),[121,p([1,h])]],[13,[131,ack0]],[14],[15]]

Result = [1,[12,ack1,p([2,a]),da(12),sa(13),[121,p([1,h])]],[13,[131,ack0]],[14],[15]]

Clock = 7

Membrane = [1,[12,ack1,p([2,a]),da(12),sa(13),[121,p([1,h])]],[13,[131,ack0]],[14],[15]]

RULE = [12,4][ack1,p([2,a]),da(12),sa(13)] → [in(13),ack1]

OLD = [1,[12,ack1,p([2,a]),da(12),sa(13),[121,p([1,h])]],[13,[131,ack0]],[14],[15]]

SUCCEDED ! New=[1,[12,p([2,a]),[121,p([1,h])]],[13,ack1,[131,ack0]],[14],[15]]

RULE = [12,7][p([2,a]) → [in(121),p([2,a])]
 OLD = [1,[12,p([2,a]),[121,p([1,h])]], [13,ack1,[131,ack0]], [14],[15]]
 SUCCEEDED ! New=[1,[12,[121,p([1,h]),p([2,a])]], [13,ack1,[131,ack0]], [14],[15]]
 Result = [1,[12,[121,p([1,h]),p([2,a])]], [13,ack1,[131,ack0]], [14],[15]]

Clock = 8

Membrane = [1,[12,[121,p([1,h]),p([2,a])]], [13,ack1,[131,ack0]], [14],[15]]
 RULE = [13,5][ack1 → [accept]
 OLD = [1,[12,[121,p([1,h]),p([2,a])]], [13,ack1,[131,ack0]], [14],[15]]
 SUCCEEDED ! New=[1,[12,[121,p([1,h]),p([2,a])]], [13,accept,[131,ack0,ack1]], [14],[15]]
 Result = [1,[12,[121,p([1,h]),p([2,a])]], [13,accept,[131,ack0,ack1]], [14],[15]]

Clock = 9

Membrane = [1,[12,[121,p([1,h]),p([2,a])]], [13,accept,[131,ack0,ack1]], [14],[15]]
 RULE = [13,2][accept → []
 OLD = [1,[12,[121,p([1,h]),p([2,a])]], [13,accept,[131,ack0,ack1]], [14],[15]]
 SUCCEEDED ! New=[1,[12,[121,p([1,h]),p([2,a])]], [13,[131,ack0,ack1]], [14],[15]]
 Result = [1,[12,[121,p([1,h]),p([2,a])]], [13,[131,ack0,ack1]], [14],[15]]

Clock = 10

Membrane = [1,[12,[121,p([1,h]),p([2,a])]], [13,[131,ack0,ack1]], [14],[15]]
 Result = [1,[12,[121,p([1,h]),p([2,a])]], [13,[131,ack0,ack1]], [14],[15]]

yes
 — ?-

6 Conclusion

In this paper we have understood the new paradigm of computation by Gheorghe Paun called “Computing with Membranes”, and we have selected protocols as an application of it and implemented in the simulation environment written in Prolog. and checked for strong simulation of each other, which proved to be 'NOT'.

References

- [1] Gheorghe Paun. Computing with membranes. *Turku Centre for Computer Science, TUCS Technical Report No 208, ISBN 952-12-0303-X, ISSN 1239-1891*, November 1998.
- [2] Mihaela Malita. Membrane computing in prolog. *Pre Proceedings of the Workshop on Multiset Processing, Curtea de Arges, Pages 159 - 175*, August 21 - 25, 2000.
- [3] Prof.Horst Reichel. Formal models of concurrency. Technical report, Theoretical Computer Science, Dresden University of Technology, September 2000.
- [4] *Data And Computer Communications, Fifth Edition, ISBN 0-13-571274-2*. Prentice-Hall, International Inc., 1997.
- [5] *Communicating and mobile Systems:the pi-Calculus, ISBN 0 521 64320 1, University of Cambridge*. Cambridge University Press, 1999.
- [6] Gheorghe Paun Jrgen Dassow. On the power of membrane computing. Technical report, Turku Centre for Computer Science, TUCS Technical Report No 217, ISBN 952-12-0330-7, ISSN 1239-1891, November 1998.