*Journal of Universal Computer Science, vol. 5, no. 9 (1999), 588-598*
*submitted: 10/6/99, accepted: 15/7/99, appeared: 28/9/99* 

# Mobile Ambients and P-Systems

Ion Petre
Turku Centre for Computer Science and
Department of Mathematics, University of Turku, Finland
ipetre@cs.utu.fi

Luigia Petre
Turku Centre for Computer Science and
Department of Computer Science, Åbo Akademi University, Finland
lpetre@abo.fi

**Abstract:** The ambient calculus and the P-systems are models developed in different areas of computer science. Still, they are based on similar concepts and structures and are inspired from the same *natural* model of computation [BeBo92]. On this basis, we point out how to transfer ideas and results from one framework to the other. We prove that any P-system can be simulated in ambient calculus. We also introduce the notion of mobile P-systems, suitable to model and motivate security features for membrane computing.
**Key Words:** Ambient calculus, P-systems, Mobility, Secure communications

## 1 Introduction

The notions of *membrane* and *membrane structure* first appeared in a paper of Berry and Boudol [BeBo92], where they were introduced as a new type of abstract machine: *the chemical abstract machine*, intended to model concurrent computations. The idea is to use a system like a chemical solution in which swimming molecules can interact with each other, according to a given set of structural behaviour laws. When certain conditions hold, the molecules are allowed to break into simpler molecules or conversely, to form more complex molecules. As pointed out in [BeBo92], this model proved to be suitable to implement mobile processes.

The idea of a machine based on a membrane structure proved to be fruitful, as it consisted in the starting point for several different models in computer science. We are particularly interested here in two such models: the ambient calculus and the P-systems. The ambient calculus was recently introduced in [CaGo98] as an abstract model attempting to capture fundamental features of distribution and mobility over wide area networks such as Internet. The central notion in this framework is the *mobile ambient* which is a named, bounded place where computation happens and which is itself mobile. Inside the ambient there can be computations taking place in parallel and also other subambients. An ambient can move with its entire internal structure outside and inside other ambients, can open (dissolve) other ambients, and can be opened, all these features being enforced by local capabilities. In this way, one obtains a framework for modelling wide area networks, in which every node is seen as an ambient, with mobile subnodes, and processes running locally. It is particularly interesting that notions like mobility and security, quite important for an Internet model, are very well suited within the ambient calculus.

The P-systems have been introduced in [Pa98] as a distributed and parallel computability model with motivations from *molecular computing* and based on the same notion of membrane structure. In each membrane there are objects placed, evolving according to some local, prioritised rules. The objects are allowed to pass through membranes and to dissolve the membrane in which they are placed at that moment. The output of the system is considered to be the collection of objects entering one special output membrane (as in [Pa98]) or alternatively, the collection of objects leaving the system. Several variants of the basic model have already been considered, introducing ideas from DNA-computing, biochemistry, etc. The model seems to be quite powerful: it is proved in [Pa99c] that the P-systems can be used to solve NP-complete problems in polynomial time.

The goal of this paper is to investigate the link between these models coming from two different fields of computer science: the *formal methods* area and the *formal languages* area. We initiate a comparative analysis between the ambient calculus and the P-systems, based on their similar structure and common concepts. We show that any P-system can be expressed in ambient calculus by a mobile ambient having the same structure. We also define the notion of *mobile P-system*, well suited to model secure communications as in the case of mobile ambients. Using this, a type system for the cells of a P-system can also be introduced to ensure the soundness of communication between different cells of a P-system.

## 2 Ambient Calculus

The Ambient Calculus is a process calculus aimed to capture the structure of dynamic networks and the behaviour of mobile computations. The calculus successfully describes the movement of processes and devices, including movement through administrative domains. A minimal version of this calculus handles only mobility primitives and proves to be, in this restricted form, computationally complete [CaGo98]. We use and present here this restricted form of the calculus, since it already exhibits the same structural model as the P-systems. We describe informally the semantics of the calculus, using the reduction '$P \to Q$' to denote the evolution of the process '$P$' into the new process '$Q$'.

The elementary structure this calculus relies on is the notion of an *ambient*, which is roughly a bounded place where computation happens. The border of an ambient is identified by a *name*. If an ambient named $n$ is bounding the computation described by a process $P$, this is expressed as $n[P]$. In general, the computation can be a parallel composition of many processes. An ambient can also contain other subambients, each with its own computation inside. Thus, the general form of an ambient is:

$$n[P_1 \mid \cdots \mid P_p \mid m_1[\ldots] \mid \cdots \mid m_q[\ldots]],$$

with $P_1, \ldots, P_p$ processes running in parallel, and $m_1[\ldots], \ldots, m_q[\ldots]$ subambients carrying in parallel their own computations. Given this form of the ambient, a real system is usually modelled by a hierarchical collection of ambients.

The structure of the processes running in an ambient is described by the following primitives: the *restriction, inactivity, parallel composition* and *replication*. They have the same semantics as in any process calculus: '$(\nu n)P$' means the creation of a fresh name $n$, restricted to the scope $P$, '$\mathbf{0}$' is the process doing nothing, '$P \mid Q$' is the parallel composition of two processes '$P$' and '$Q$', and '$!P$' denotes the unbounded replication of the process '$P$', being equivalent to '$P \mid !P$'.

In addition, there are primitives describing the dynamics of ambients hierarchy. The evolution of this tree structure is restricted by *capabilities*: an ambient willing to move upwards or downwards in the hierarchy must have the permission to do so. The capabilities to enter or to exit an ambient (mobility capabilities) can be transmitted as values, using communication primitives which we do not describe here but can be found in [CaGo98], or can be deducted from the name of that ambient (for security reasons, in ambient calculus, the names of the ambients are thought of to be secret, but they can also be subject to communications). Also, the operation of opening (dissolvation of) an ambient's border is controlled by capabilities.

A process '$M.P$' executes an action regulated by the capability '$M$' (which implies the consumption of '$M$'), and then continues with '$P$'. There are three kinds of capabilities, and each of them reduces '$M.P$' in its specific way, as we describe in the following.

An *entry* capability '$in\ m$' is used to enter into an ambient named '$m$'. It can be used in an action '$in\ m.P$' that enables the surrounding ambient of the action, say '$n$', to enter into a sibling ambient '$m$'. The action is performed when such an ambient exists 'nearby', i.e. both '$m$' and '$n$' are the children of the same parent ambient. When there is no ambient '$m$' nearby '$n$', the action is blocked; when several ambients named '$m$' exist nearby, one of them is non-deterministically chosen and entered. Hence, the following reduction formally describes the entry capability:

$$n[in\ m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R].$$

An *exit* capability '*out* $m$' is used to exit from an ambient named '$m$'. It can be used in an action '*out* $m.P$' that enables the surrounding ambient of the action, say '$n$', to exit from its parent ambient '$m$'. The action is performed when such an ambient exists i.e., '$n$' is the child of '$m$'. When the parent ambient is not named '$m$', the action is blocked. This is formally described in the reduction

$$m[n[\mathit{out}\ m.P \mid Q] \mid R] \rightarrow n[P \mid Q] \mid m[R].$$

An *open* capability '*open* $m$' is used to dissolve an ambient named '$m$'. It can be used in an action '*open* $m.P$' that enables the dissolving of a sibling ambient '$m$'. The action is performed when such an ambient exists 'nearby', i.e. '$m$' and '*open* $m.P$' belong to the same parent ambient, say $n$. When there is no ambient '$m$' nearby '*open* $m.P$', then the action is blocked; when many ambients named '$m$' exist nearby, one of them is non-deterministically chosen and dissolved. The following reduction formally describes the open capability:

$$n[\mathit{open}\ m.P \mid m[Q]] \rightarrow n[P \mid Q].$$

These primitives are enough to encode more complex operations as renaming of ambients, detecting the presence of a certain ambient, auto-dissolvation (the process *acid* which we will use later on), iteration of processes, and even protocols of authentication (for firewalls crossings) and secure communications [Ca99, CaGo98].

Since the ambient calculus formalises the structure and the properties of mobile computations within wide area networks the main interest in this field is the study of new programming methodologies and languages for global computations, based on the principles of ambients. There are also approaches to extend the calculus by considering diverse types for ambients [CaGo99, CaGhGo99]. More theoretical aspects regarding the semantics of ambients are treated by Gordon, Cardelli in [GoCa99].

## 3 P-systems

The *P-systems* have been recently introduced in [Pa98] as a biologically motivated model for distributed parallel computing. Such a system is based on a hierarchically arranged, finite *cell-structure*. Each cell is delimited by a *membrane* and can contain several other cells and some objects, swimming in it. These objects evolve according to some given prioritised evolution rules, can pass through a membrane to go to an adjacent cell, or can dissolve the membrane of their current cell. In the case of this last event, all the objects in the former membrane remain in the immediately superior cell and they will further evolve according to this one's evolution rules. The rules of the dissolved cell are lost. The objects can evolve in dependence to each other (cooperative systems) or independently (non cooperative systems).

More formally, consider a cell and an evolutionary rule $r$ of it,

$$r : x_1 \dots x_k \longrightarrow (y_1 \dots y_l, here)(z_1, in_{i_1}) \dots (z_m, in_{i_m})(u_1 \dots u_p, out)\delta,$$

with $k \geq 1$, $l, m, p \geq 0$, and possibly without the final $\delta$ (in this case we say that $r$ is a non-dissolving rule). If at some point of the computation, we have in the cell some (possibly more than one) objects labelled by $x_1, \dots, x_k$, then one object from each type may evolve according to $r$. They will be erased from the cell, and instead of them, the objects $y_1, \dots, y_l$ will be created in the same cell, other objects $u_1, \dots, u_p$ will be created and sent outside the cell, and the objects $z_1, \dots, z_m$ will be created and sent to the cells $i_1, \dots, i_m$, respectively. Also, if $r$ is a dissolving rule, the cell is dissolved, its rules are lost, and its content is kept by the parent cell.

If the cells $i_1, \dots, i_m$ in which the new objects $z_1, \dots, z_m$ must enter according to $r$ do not exist inthe current cell, then the rule is not applicable and the objects $x_1, \dots, x_k$ may evolve according to some other rule. The rule is also not applicable in the current step of the computation, if some other rule with higher priority is applicable. Finally, if an object can evolve according to more than one rule, then the rule to be applied is chosen non-deterministically.

Such a system evolves in parallel: all the cells evolve in the same time and moreover, all the objects inside a cell evolve in parallel. In this way, a computing device is obtained: we start from an initial configuration, defined by a cell-structure with some initial content in each cell and let the system to evolve until it halts. If the P-system does not halt at all, we have an unsuccessful computation. During the evolution of the system, we observe it from outside and we build a string by sequentially arranging the objects in the order they leave the system (the skin cell). When several objects leave the system at the same time, any ordering of them will be accepted. The set of finite words obtained in this way forms a language which is said to be generated by the P-system.

Many variants have already been considered. In [DaPa99], one considers variants of P-systems with and without cooperative rules, prioritised evolutionary rules, and membranes dissolving, studying the computing power of each of them. Also, in [Pe99] a normal form for one of these variants is given, proving that any non cooperative P-system can be simulated by a P-system of depth 2.

Connections to DNA-computing are considered in [Pa98, Pa99b], where the prioritised evolution rules are replaced by splicing rules. Thus, a new model of membrane computing is obtained and further studied in [PaYo99a] and [PaYo99b]. P-systems, with the underlying membrane structure organised as a graph rather than a tree, are discussed in [PaSaYo99]. Furthermore, some other ideas from *natural computing* are introduced in [Pa99a], where one eliminates the labelling of the cells and the priority relations among the evolution rules. Instead of them, electrical charges for membranes and objects, plus a variable permeability of the cells are considered.

Generalised P-systems, able to transfer objects and operators between cells, are defined and studied in [Fr99]. Also, P-systems with a different notion of membrane structure are studied in [At99]. The division of membranes is taken into consideration in [Pa99c]. In this framework it is proved an important result showing that the SAT-problem (well known to be NP-complete) can be solved in linear time using this type of P-systems.

There are many open problems in this rapidly growing area. Other ideas from biochemistry and natural computing can be further introduced and studied in the framework. There are also problems related to the structural complexity of these models and to decidability questions. Moreover, there are basic tools like necessary conditions, characterisations, and normal forms for some of the variants, which are needed in this framework.

## 4 Expressing P-systems in ambient calculus

We investigate in this section the expressibility of P-systems in ambient calculus. Although the approach adopted in the following can be further refined for the general case, we consider here for simplicity the non cooperative P-systems only. Ideas on how to model some other features considered in variants of P-systems are, however, given in the end of this section.

Let us consider a P-system $\Pi$, which we can assume that it has the working alphabet distinct from the set of labels of its cells. We construct an ambient $\Lambda$ to simulate $\Pi$ in ambient calculus. The ambient $\Lambda$ has basically the same structure as $\Pi$: the same nested sequence of membranes, with the same names. Still, inside $\Lambda$, we create some other ambients to model the objects evolving in $\Pi$. They are empty ambients named as the objects that they stand for, and they are manipulated throughout the calculus, similarly as in $\Pi$. Obviously, we can arrange to have in the beginning of the simulation an initialisation phase, in which each ambient creates its local data (the empty ambients), according to the initial content of the corresponding cell of $\Pi$.

Consider then a cell $i$ of $\Pi$ and an arbitrary rule $r$ of $i$,

$$r : x \to (x_1, \dots x_m, \mathit{here})(y_1 \dots y_n, \mathit{out})(z_1, \mathit{in}_{k_1}) \dots (z_p, \mathit{in}_{k_p})\delta,$$

with $m \geq 0$, $n \geq 0$, $p \geq 0$, and possibly without the final $\delta$. For this rule, we consider the following expression in ambient calculus:

$$E_r = (x, k_1, \dots, k_p \Rightarrow P_r),$$

saying that in the presence of ambients $x, k_1, \dots, k_p$ in the local ambient, the process $P_r$ will be allowed to run (we will call it active). These are precisely the conditions in which the rule $r$ is applicable, ignoring for the moment the priority relations among rules. The expression $E_r$ is a syntactically richer expression than the ambient calculus allows (only $P_r$ would have sufficed), but we explicitly specified which ambients should be present for the rule to be enabled for the sake of clarity. The process $P_r$ simulates the rule $r$ in the ambient $i$ of $\Lambda$ as follows:

$$
\begin{aligned}
P_r =!(\ open\ x.\ (\ (\ & x_1[\,] \mid \cdots \mid x_m[\,]) \mid \\
& (\ y_1[\,].mv\ out_i\ y_1 \mid \cdots \mid y_n[\,].mv\ out_i\ y_n\ ) \mid \\
& (\ z_1[\,].mv\ in_{k_1}\ z_1 \mid \cdots \mid z_p[\,].mv\ in_{k_p}\ z_p\ ) \mid \\
& (\ acid)\ \ )\ ),
\end{aligned}
$$

where the process *acid* is in $P_r$ only if $r$ contains the dissolvation symbol $\delta$. The effect of $P_r$ is that for each ambient named $x$, the following procedure is launched: the ambient $x$ is opened and then, in parallel, the ambients $x_1, \ldots, x_m$ are created in the ambient $i$, the new ambients $y_1, \ldots, y_n$ are sent outside the ambient $i$, and the new ambients $z_1, \ldots, z_p$ are sent to their destinations $k_1, \ldots, k_p$ according to the rule $r$ of $\Pi$. Finally, the ambient $i$ is opened if $r$ is a dissolving rule. Obviously, there may be several processes $P_{r_1}, \ldots, P_{r_j}$ attempting to open the same ambient $x$. In this case, the operator *open* succeeds for only one process, chosen randomly among $P_{r_1}, \ldots, P_{r_j}$ and the other processes are cancelled. This is in accordance to the principles of P-systems: if an object $x$ can evolve according to several rules, the system will randomly choose the rule to be applied to $x$. Hence, the result of the process $P_r$ is precisely the same as the effect of the rule $r$ on $\Pi$.

We now need an operation on ambients to simulate a prioritised choice. We consider here a *prioritised choice operator* $\oplus$, similar to the non-deterministic choice operator $+$, already introduced in the calculus. The result of

$$n \Rightarrow P\ \oplus\ m \Rightarrow Q$$

is that $P$ will be executed with priority over $Q$, provided that an ambient $n$ exists. If this is not the case, then $Q$ can be executed if an ambient $m$ exists. We define this operator in two phases. In the first one, the ambients $tn$ or $fn$ and $tm$ or $fm$ are created, standing for boolean flags: $tn$ is created if ambient $n$ exists, otherwise $fn$ is created and similarly for the ambient $m$. Having these flags, in the second phase we use the choice operator to decide which one of $P$ and $Q$ is to be executed. Our operator is thus:

$$
\begin{aligned}
&((in\ n.\ out\ n.\ tn[\,].\ open\ fn) \mid fn[\,] \mid \\
&(in\ m.\ out\ m.\ tm[\,].\ open\ fm) \mid fm[\,]). \\
&(tn \Rightarrow P.\ open\ tn.(tm \Rightarrow open\ tm\ +\ fm \Rightarrow open\ fm) \\
&+ \\
&fn \Rightarrow open\ fn.(tm \Rightarrow Q.\ open\ tm\ +\ fm \Rightarrow open\ fm))
\end{aligned}
$$

This operator is easily extensible to a finite prioritised choice. The result of an operation

$$m_1 \Rightarrow P_1 \oplus m_2 \Rightarrow P_2 \oplus \cdots \oplus m_k \Rightarrow P_k$$

is that the process to be executed will be $P_l$, where $l$ is minimal, $1 \leq l \leq k$, such that there exists an ambient named $m_l$ in the ambient $i$ of $\Lambda$. In other words, the process $P_l$ will be executed with higher priority than the processes $P_{l+1}, \ldots, P_k$, provided that there is an ambient $m_l$ in $i$.

Assume now that the priorities among the rules of the cell $i$ are

$$\begin{gathered} r_{11} > r_{12} > \cdots > r_{1l_1}, \\ \vdots \\ r_{k1} > r_{k2} > \cdots > r_{kl_k}, \end{gathered}$$

with $l_1, \ldots, l_k \geq 1$. Then, the process running in the ambient $i$ is constructed as follows:

$$\begin{aligned} P_i = (rec\ \mathrm{T})\ (\ (\nu r) r[out\ i.in\ i.open\ r]\ . \\ (\ E_{r_{11}} \oplus \cdots \oplus E_{r_{1l_1}} \mid \cdots \mid E_{r_{k1}} \oplus \cdots \oplus E_{r_{kl_k}}) \mid \mathrm{T}\ ). \end{aligned}$$

The first part of the recursion in $P_i$ ensures that the process will run only as long as the ambient $i$ is not opened (in P-systems the rules are lost, once their home cell is dissolved). Then, in parallel for each priority chain, the active process with the highest priority is chosen. The procedure is then iterated in parallel in each ambient of the system thus, fully simulating the computation of $\Pi$.

There are features considered in some variants of P-systems, which can be easily simulated in ambient calculus. We give here two ideas. The P-systems allowing the division of cells can be easily modelled in ambient calculus by creating new ambients with the same name. Also the variant in which both the objects and the membranes have no names, but they are provided with electrical charge, can be modelled by an ambient for which all the subambients are named either *pos* or *neg*, depending on the charge of the corresponding membrane. One can even change the name of an ambient during the computation, simulating a change in the electrical charge of a cell.

## 5 Mobile P-systems

As we already mentioned, a P-system is a distributed parallel system. The communications in this model are possible only between adjacent cells. Thus, two non-adjacent cells willing to communicate would have to send the message step by step through the path between them, with the obvious danger for the message to be detoured or modified by a third party. Hence, as in any distributed model, one needs to have secure communications between any pair of components. We propose in this section a model to accomplish this, importing an idea from the mobile ambients to P-systems.

Perhaps the most important difference between an ambient and a P-system is that the former has a dynamic structure while the latter has a rather static one. The mobility of the ambients and the capabilities required to perform an action are the ingredients that enable the security features in ambient calculus. We prove here that it is enough to consider a restricted form of mobility for P-systems in order to obtain similar features for membrane computing.

We give the following definition. A *mobile P-system* working on the finite alphabet $\Sigma$ is a P-system in which new cells of a special type can be created during the computation, using rules of the form

$$\alpha \to \beta\nu(\ key,\ k_o, k_1, \ldots, k_n),$$

where $key, k_o, k_1, \ldots, k_n \in \Sigma$, and $\alpha \to \beta$ is an evolutionary rule as described in section 3. The effect of this rule is the following: if $\alpha$ is available in the current cell, then a new cell is created and some new objects $k_o, k_1, \ldots, k_n$ are created inside this cell. We attach to it a label $T_{i,k_n}$, where $i$ is the label of the current cell, and we call it *travelling cell* from $i$ to $k_n$.

The travelling cell created above will have by definition the following rules:

$$\begin{array}{ll} d & : key \to \delta, \\ i_1 & : k_1 \to \mathbf{move}(in_{k_1}), \\ i_2 & : k_2 \to \mathbf{move}(in_{k_2}), \\ \vdots & \\ i_m & : k_n \to \mathbf{move}(in_{k_n}), \\ o & : k_o \to k_o\mathbf{move}(out), \end{array} \tag{1}$$

with the priority relations: $d \geq i_1 \geq i_2 \geq \cdots \geq i_m \geq o$. The intended meaning of a rule $k_i \to \mathbf{move}(in_{k_i})$ is that whenever the cell labelled by $k_i$ is a sibling of the travelling cell, then the latter one will enter the cell $k_i$. Similarly, the effect of the rule $k_o \to k_o\mathbf{move}(\mathrm{out})$ is that the travelling cell will move outside its current surrounding cell, whenever none of $k_1, \ldots, k_n$ is nearby. We say that the movement of the travelling cell created above is controlled by $k_1, \ldots, k_n$. The travelling cell will know that it arrived at the proper destination when its parent cell will introduce in it an object *key*. Then, according to the rule $d$ above, the travelling cell will be dissolved, releasing its content at the intended destination.

Using the travelling cells, any two 'static' cells will be able to establish a direct secure communication provided that they share a common *key*. The main question here is how to know the current path from a cell to another, since this may change throughout the computation, due to the dissolvation of some cells.

Consider a cell $j$ in a mobile P-system $\Pi$, and the path from the skin cell to $j$,

$$0 \to j_1 \to \cdots \to j_{m-1} \to j,$$

at the starting point of the computation. It is clear from the above definition that a travelling cell with the movement controlled by $j_1, \ldots, j_{m-1}, j$, will eventually arrive in the cell $j$, regardless of how many of the intermediate cells $j_1, \ldots, j_{m-1}$ are still alive in the current moment of the evolution. If the destination cell $j$ has been dissolved, then the travelling cell will arrive and remain in the skin cell, which is all right: the communication fails since one of the parties has been destroyed.

The security feature relies in fact on the dissolving *key* of the travelling cell. Any cell aiming to break the carrier and to find out its content, must introduce a possible key into the carrier. An important observation here is that the movement of the travelling cell cannot be controlled by anybody else excepting itself, and so, any 'evil' cell has only one chance to break the carrier. Moreover, the name of the travelling cell can be thought of to be secretly shared by the sender and the destination, thus making the task of breaking the communication system even more difficult. A protocol that changes the key for every pair of communicating cells can be easily imagined and in this way, a perfectly secure communication mechanism is obtained. As a consequence, one can think that in this model the direct communications between any two cells are possible, thus gaining more flexibility. Also the type system in which one can specify a cell to be either static or mobile, similar as in the case of mobility types for ambients (see [CaGhGo99]), seems to help the construction of more involved examples and techniques within P-systems.

## 6 Conclusions

It is very interesting that the natural concept of membrane structure is capable to model and simulate both the organisation of a complex network, and the processes running in that network. This is indeed possible: one can use the mobile ambients to describe a dynamic network, fully capturing its mobility and security features, and one can use the P-systems to describe the processes running in the network, using the same model of membrane structure. Thus, the Web-computing can be thought of uniformly, both at the structural level, and at the level of processes running in the network, based on a natural concept of biochemical inspiration.

Although the goals and the motivations of ambient calculus and P-systems are different, we showed that the transfer of ideas and results from one model to the other is indeed possible. It seems like an interesting research direction to further investigate possible connections between these frameworks.

The mobile P-systems offer more flexibility than the basic variant, and they could be used for more involved cryptographic protocols within P-systems. Their computational power, descriptional complexity, and other properties still remain to be investigated.

**Aknowledgement**

The work of the first author was supported by the Academy of Finland, under grant 44087.

## References

[At99] A. Atanasiu, About a class of P-systems, submitted.
[BeBo92] G. Berry, G. Boudol, The chemical abstract machine, Theoret. Comp. Sci. 96 (1992), 217–248.
[Ca99] L. Cardelli, Abstractions for mobile computation, in *Secure internet programming*, (J. Vitek, Ch. Jensen, eds.), LNCS 1603, 1999.
[CaGo98] L. Cardelli, A. Gordon, Mobile ambients, in *Proceedings of FoSSaCS'98* (M. Nivat, ed.), LNCS 1378, 140–155.
[CaGo99] L. Cardelli, A. Gordon, Types for mobile ambients, in *Proceedings of POPL'99*, ACM, 79–92.
[CaGhGo99] L. Cardelli, G. Ghelli, A. Gordon, Mobility types for mobile ambients, in *Proceedings of ICALP 1999*, to appear.
[DaPa99] J. Dassow, Gh. Păun, On the power of membrane computing, J. of Universal Computer Sci., 5, 2 (1999), 33–49.
[Fr99] R. Freund, Generalized P-systems, *Proc. of FCT'99* (G. Ciobanu, Gh. Păun, eds.), Lecture Notes in Computer Sience, 1684, Springer-Verlag, 1999.
[GoCa99] A. Gordon, L. Cardelli, Equational properties of mobile ambients, in *Proceedings of FoSSaCS'99*, to appear.
[Pa98] Gh. Păun, Computing with membranes, summitted. Also as Turku Centre for Computer Science Report No 208, 1998 (www.tucs.fi).
[Pa99a] Gh. Păun, Computing with membranes: a variant, submitted. Also as Auckland University, CDMTCS Report No 098, 1999 (www.cs.auckland.ac.nz/CDMTCS).
[Pa99b] Gh. Păun, Computing with membranes. An introduction, Bulletin of the EATCS, 67 (1999), 139–152.
[Pa99c] Gh. Păun, P Systems with Active Membranes: Attacking NP Complete Problems, submitted. Also as Auckland University, CDMTCS Report, 1999 (www.cs.auckland.ac.nz/CDMTCS).
[PaRoSa98] Gh. Păun, G. Rozenberg, A. Salomaa, Membrane computing with external output, submitted. Also as Turku Centre for Computer Science-TUCS Report No 218, 1998 (www.tucs.fi).
[PaSaYo99] Gh. Păun, Y. Sakakibara, T. Yokomori, P systems on graphs of restricted forms, submitted, 1999.
[PaYo99a] Gh. Păun, T. Yokomori, Membrane computing based on splicing,*Preliminary Proc. of Fifth Intern. Meeting on DNA Based Computers* (E. Winfree, D. Gifford, eds.), MIT, 1999, 213–217.
[PaYo99b] Gh. Păun, T. Yokomori, Simulating H systems by P systems, *Journal of Universal Computer Science*, 5 (1999), to appear.
[PaYu99] Gh. Păun, S. Yu, On synchronization in P systems, *Fundamenta Informaticae*, 38, 4 (1999), 397-410.
[Pe99] I. Petre, A normal form for P systems, *Bulletin of the EATCS*, 67 (1999), 165-172.