Formal Modelling for In-silico Experiments with Social Insect Colonies

Ioanna Stamatopoulou[1], Ilias Sakellariou[2], Petros Kefalas[2], George Eleftherakis[2]

[1]SEERC, 17 Mitropoleos Str, Thessaloniki, Greece, istamatopoulou@seerc.org
[2]City College, Dept. of Computer Science, 13 Tsimiski Str, 54624 Thessaloniki, Greece, {i.sakellariou, kefalas, eleftherakis}@city.academic.gr

Abstract

Social insect colonies present an interesting problem for formal modelling due to their outstanding characteristics, such as self-organisation and emergence. In this paper, we experiment with two different formal methods, Communicating X-machines and Population P Systems, which can be used separately to model biologically inspired. systems. The case of Pharaoh's ants is used as a vehicle of study. We discuss the advantages of each method and we present a framework that leads to a rapid implementation and simulation of such multi-agent systems.

1 Introduction

Study of social insects' colonies, such as ants and bees, reveal the need for computational models, which are able to handle the highly dynamic structure of any biological or artificial system that exhibits emergent behaviour. Such computational models would facilitate understanding of self-organisation phenomena that appear in those colonies. Part of bio-informatics technology aims at developing in-silico models and simulations that will complement in-vitro and in-vivo biological experiments. On the other hand, knowledge gained from observation in these experiments could be used to develop artificial multi-agent systems in which simple components with simple interactions will achieve a complex overall behaviour.

A social insect colony is an example of a multi-agent system where the capabilities of the entire colony are much greater than that of any individual. Members of a colony have a distinct role that determines what behaviour they must demonstrate. For example, in ant colonies we may find foraging ants, workers, queens etc. The behaviours of the social insects are directed towards the benefit of the colony as a whole and this is done through self-organisation. This is achieved through local interactions with other insects and the environment, since no insect has a global view of the environment. The major challenge for social insect research is the integration of individual behaviours, thereby understanding the emergent colony-level behaviour.

Various aspects of ant colonies have been modelled using different approaches in order to explain some behavioural characteristics or to ease simulating different conditions that might restrict or influence their behaviour. In this paper we present and compare two models by applying them to a case study involving the simulation of the behaviour of an ant colony, Pharaoh's ants in particular, inside their hive.

Monomorium pharaonis, also called the Pharaoh's ants, is a species of ants that originated from North Africa. A typical colony comprises of 100 to 5000 ants and contains a queen, a number of workers, pupae and some brood. A number of about 100 to 200 ants is adequate to study their behaviour in-vitro. Inside the nest, the ants are inactive for most of their time. An ant becomes active if it becomes hungry or if it is being recruited by another ant.

The assumptions that are made for this study are: (a) the colony is situated in a rectangular environment and only consists of workers, (b) the ants are either inactive or move around looking for food, (c) when two ants meet they might share food, if one is hungry and the other has food supplies, and (d) the ants go out to forage when they are hungry, no food source is identified and a pheromone trail leading to an exit of the nest is discovered.

Although fairly simple the above is a realistic case study and of interest since it shows a combination of independent ant behaviours as well as synchronised behaviour, in the case that two ants come across to exchange food. It also has an important degree of repetitiveness using the same type of ant in a number of instances but also with slight variations between them (through the food distribution across the colony, different ant positions in the environment, different individuals hunger thresholds etc.). Finally, it exhibits aspects of self-organisation of the ant colony.

From a modelling perspective, there are several interesting properties of the system that are challenging. The individual behaviour of the ants must be modelled enabling them to perceive their environment. The ant population is not static (new ants enter the nest, some leave, others die of hunger etc.) but neither is their communication network (pairs of ants communicate under particular conditions). Overall, the configuration of the colony is highly dynamic and constantly changes over time.

In the next section, we show how two different formal methods can be employed for modelling social insect behaviour and we investigate their suitability for modelling the behaviour of Pharaoh's ants. In Section 3, we discuss the steps towards a rapid simulation of the formally modelled behaviour using the NetLogo platform. Finally, in Section 4 we discuss our findings from experimenting with the two formal methods and the simulation and Section 5 concludes this paper.

2 Formal Modelling

Formal methods are based on rigorous mathematical notations, which aim at describing systems in the early stages of software development. Such formal descriptions are useful for precisely specifying a system's data and/or control and, as a consequence, for verifying whether certain properties are true (through formal verification or model checking) as well as to test whether the final product meets the initial description (through complete formal testing). All of these stages are crucial in the development process and researchers involved with formal methods claim that "correct" software can only be achieved through the use of formal methods.

A variety of formal methods exist, each one holding prominent characteristics that make it suitable for modelling different classes of problems. In our case, we are going to use two different formal methods, namely Communicating X-machines (CXM) and Population P Systems with active cells (PPS), which are found to hold characteristics that are well suited to the problem in question. In the following, we do not extensively present and discuss formal definitions, which can be found elsewhere. Instead, we focus on the modelling strengths of each method, presenting them in a rather informal way, and elaborate with mathematical notation and its meaning where appropriate.

2.1 Communicating X-machines

X-machines (XM) were firstly introduced by Eilenberg [1974] based on the idea that a finite state machine is extended with a structure that represents its memory, while a stream of inputs trigger the appropriate functions that annotate the transitions between its states. The method greatly facilitates the software engineering process of specifying or modelling reactive systems [Holcombe, 1998]. XM models can be specified in any mathematical notation but also in an appropriately defined notation called XMDL (X-machine Description Language) [Kefalas et al, 2003b]. The latter initiated the development of various tools, which aid the modelling process as well as the animation, testing and model checking of XM models. Communicating X-machines are an extension of the XM method, which deals with the development of large scale systems that consist of several interacting components, each of which is described as an XM [Kefalas et al, 2003a].

For the problem at hand, one initially has to create a model for an ant, starting by identifying the number of states. There are five states which the ant can be in: (a) *inactive*, a non-hungry ant holding food, (b) *hungry*, an ant holding a food quantity that is below its hunger threshold, (c) *giving*, a non-hungry ant that perceives a hungry one and shares its food, (d) *taking*, a hungry ant that perceives an inactive ant and receives food from it, and (e) *dead*, for an ant whose food quantity has dropped to zero. Formally we write $Q = \{inactive, hungry, giving, taking, dead\}$.

The memory of the ant holds (a) its *current position*, (b) the *amount of food*, it carries, (c) a number denoting the *food quantity threshold*, below which the ant becomes hungry, (d) the *food decay rate*, a number denoting the quantity of food that is consumed by the ant in each time unit, and (e) the *food portion*, i.e. the food amount to be given by an ant that is carrying food to another which is hungry. We choose to represent all of them as natural numbers.

The ant is modelled so that it accepts a tuple (*pos*, *stimuli*) as input to its functions. The first element of the input represents the coordinates in which stimuli are perceived, whereas the second element is the description of the stimuli. The latter can be *pheromone* or *space*, describing an empty position with or without pheromone, a hungry, *ha*, or a non-hungry, *nha*, ant, or, finally, a number greater than zero representing the food quantity that is received by a non-hungry ant that shares its food. Formally, Σ: $(N \times N) \times (\{ha, nha, pheromone, space\} \cup N)$.

The transitions between states of the XM, indicated by the diagram in Fig. 1, are the functions:
Φ = {*search*, *follow_trail*, *become_hungry*, *ignore_hungry_ant*, *die*, *do_nothing*, *meet_non_hungry_ant*, *meet_hungry_ant*, *no_food_to_give*, *become_hungry*, *take_enough_food*, *give_food*, *take_not_enough_food*}

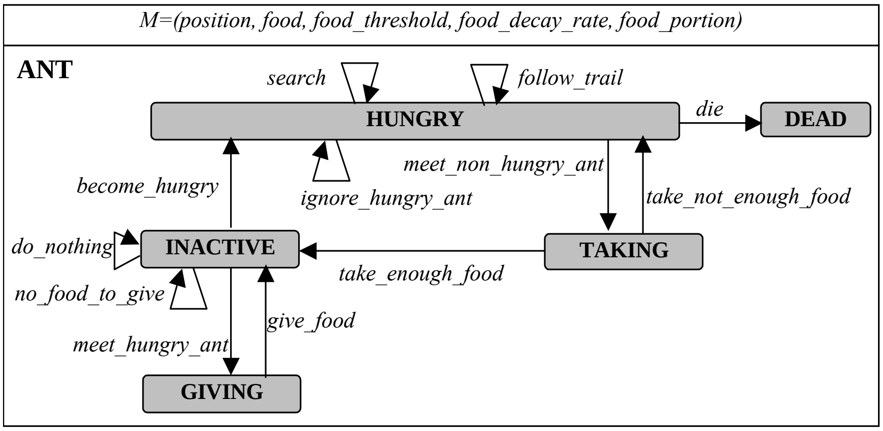


***Figure 1.** The state transition diagram of the ant*

Functions are triggered by an input and the contents of the memory, and they produce an output while updating the memory. For example, when in the *inactive* state an ant may (a) perceive a hungry ant and if the food quantity the former is carrying is enough, this will bring it to the *giving* state whereas if it does not have enough food to give, it will ignore the hungry ant and remain in the *inactive* state; (b) become hungry, if the amount of food it carries drops below its hunger threshold, or (c) do nothing; in the presence of no other stimuli the inactive ant will just consume an amount of food equal to the decay rate. The only thing that an ant in the *giving* state can do is to actually give the food to the hungry ant that has been just perceived. In formal notation:

become_hungry ((*pos*, *in*),(*myPos*, *food*, *fThreshold*, *fDecay*, *fPortion*)) =
 ((*got_hungry*), (*myPos*,*newFood*, *fThreshold*, *fDecay*, *fPortion*))
 if *newFood* $\leq$ *fThreshold*
 where *newFood* $\leftarrow$ *food*-*fDecay*

XMs can communicate by directing the output of one XM function as input to another. Figure 2 shows an instance of two ants communicating while sharing food.

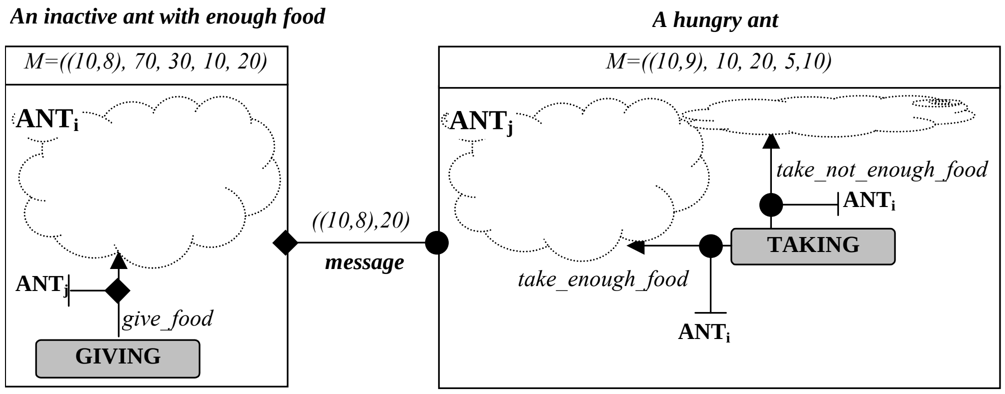


Figure 2. *Two ants communicating while sharing food. The inactive ant's function* give_food *sends as output (* ♦ *symbol) the food amount it is willing to share to be received as input (*● *symbol) by the* take_enough_food *function of the hungry ant.*

As mentioned earlier, one can verify models expressed as XMs against the requirements, since one can prove whether certain properties, implicitly defined over the memory, are true [Eleftherakis, 2003]. Additionally, in the case that the model is used as a design for a possible implementation, a test set that is guaranteed to determine its correctness may be produced, under certain well-defined conditions [Holcombe & Ipate, 1998].

2.2 Population P Systems with active cells

Membrane computing represents a new and rapidly growing research area, which is part of the natural computing paradigm [Paun, 2002; Paun et al., 2002; Martin-Vide et al., 2004] and has been introduced with the aim of defining a computing device, called a P System, which abstracts from the structure and the functioning of living cells [Paun, 2000]. The membrane structure of a cell is the hierarchical arrangement of all membranes embedded in the skin membrane that identifies distinct regions inside the P System. In the same way that each cell compartment contains its own enzymes and molecules that participate in chemical reactions, each P System region gets assigned a finite multiset of objects and a finite set of rules, either modifying the objects or moving them between regions.

A natural generalisation of the P System model can be obtained by considering P Systems whose structure is defined as an arbitrary graph, called a Population P System with active cells. Each node (*cell*) in the graph represents a membrane, which gets assigned a multiset of objects and a set of rules for modifying these objects (*transformation* rules), communicating the objects alongside the edges of the graph (*communication* rules), generating new cells (*division* rules) and destroying cells (*cell death* rules) [Paun, 2002]. The graph's edges, which denote communication channels, can change over time by applying *bond-making rules*, under certain conditions.

Our main interest lies in the ability of a PPS to reconfigure its own structure through the division, death and bond-making rules. These are rules that capture the ways in which the colony evolves through the appearance and removal of agents or communication links. For example, when two ants are close to each other and one of them is hungry and the other is inactive, a rule must create a communication link. This is shown by the following bond-making rule:
$([taking, pos_1], ant; ant, [giving, pos_2])$
    if $neighbours(pos_1, pos_2)$

The following cell death rule removes an ant from the system, when it cannot find food and its food quantity drops to zero:
$(food \rightarrow \dagger)_{ant}$, if $food \leq 0$

3 Simulation

Apart from verifying the proposed model using formal methods, the presence of a large number of XMs interacting based on spatial information dictated the need for a simulation application that would animate the execution of the proposed model. Such an animator would allow us to visualize the system's operation as well as collect statistical data concerning the colony's behaviour in the hive. Implementing such an animator requires a tool that supports simulation of a large number of autonomous entities, the Pharaoh's ants in our case and facilitates the creation of a graphical representation of the environment and the collection of experimental data. One of the best representatives of such tools is NetLogo.

NetLogo [Wilensky, 1999] is a modeling environment targeted for simulation of multi-agent systems that consist of a great number of agents. NetLogo offers a simple functional language, in which behaviours of agents can be encoded, and a programming environment that allows the easy creation of a GUI for a simulation supporting a great number of parameters. The environment is an excellent tool for rapid prototyping, initial testing of multi-agent systems and animation of the modelled system.

There are two types of agents in NetLogo: *patches* that are static agents, i.e. components of a grid on which *turtles*, i.e. agents able to move, "live" and interact. The former allow the modelling of environmental properties in a simulation, such as the existence of pheromone in a specific hive position. The latter can be used to model fully capable agents such as the ants. The programming language allows the specification of the behaviour of each patch and turtle, and of the control of the execution. Moreover each turtle can have its own set of variables and this greatly facilitated the implementation of the hive's simulation, since each ant (turtle) in the simulation maintained its own X-machine memory structure and information about its state. Monitoring and execution of the agents is controlled by an entity called *observer* that "asks" each agent to perform a specific computational task.

As stated, the motivation behind building the NetLogo simulation was to test the proposed X-machine model for the Pharaoh's ants and at the same time perform a number of experiments to investigate the colony's behaviour in a hive that contains a large number of ants. Thus, instead of building a completely ad-hoc model of the ants, based on a subjective interpretation of the functions and transitions described in the XMDL model, we have chosen to implement an X-machine meta-interpreter that will animate the specified XM model from a NetLogo representation of the XMDL specification. So that the above is better illustrated, consider the following X-machine function (defined in the X-machine Description Language):

```
#fun become_hungry ((?p, ?in), (?pos, ?f, ?ft, ?fdr, ?fp)) =
  if ?nf =< ?ft then
  ((got_hungry), (?pos, ?nf, ?ft, ?fdr, ?fp))
  where ?nf <- ?f - ?fdr.
```

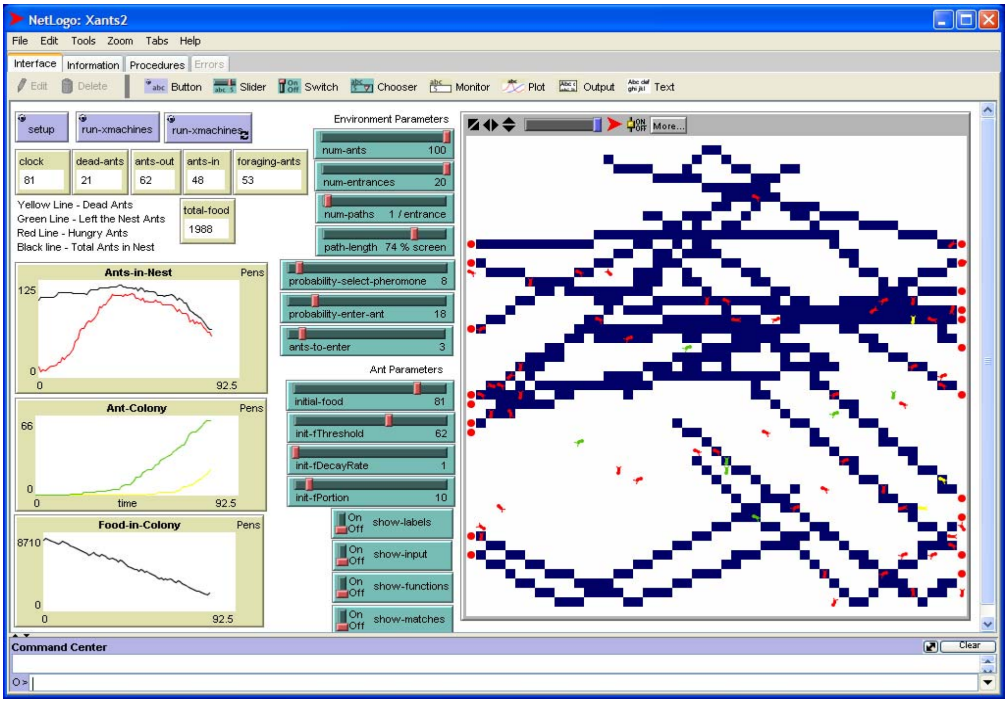
In our simulation the above is translated into the following NetLogo function:

```
to-report become_hungry [px? py? in?]
  let nf? f? - fdr?
  ifelse nf? <= ft?
   [report(list true "got_hungry" (list xcor ycor nf? ft? fdr? fp?))]
   [report(list false)]
end
```

The functions are executed by the meta-interpreter which continuously executes the following loop:

```
1. Execute Applicable Dynamic Reconfiguration Rules
2. Ask each ant to:
   2.a Determine its input based on status of its neighbouring
       positions.
   2.b Determine the applicable function based on input and state of
       the ant.
   2.c Apply changes to its memory and state specified by the
       selected function of step 2.b.
```

Implementation of the dynamic reconfiguration meta-rules is rather straightforward in NetLogo, given that the observer "controls" the execution of agents and has access to all their internal variables, i.e. a complete view of the state of computation.


NetLogo: Xants2
File Edit Tools Zoom Tabs Help
Interface Information Procedures Errors
Edit Delete Button Slider Switch Chooser Monitor Plot Output Text
setup
run-xmachines
run-xmachines
More...
Command Center
Clear
O>

clock 81
dead-ants 21
ants-out 62
ants-in 48
foraging-ants 53
total-food 1988

Yellow Line - Dead Ants
Green Line - Left the Nest Ants
Red Line - Hungry Ants
Black line - Total Ants in Nest

Environment Parameters
num-ants 100
num-entrances 20
num-paths 1 / entrance
path-length 74 % screen
probability-select-pheromone 8
probability-enter-ant 18
ants-to-enter 3

Ant Parameters
initial-food 81
init-fThreshold 62
init-fDecayRate 1
init-fPortion 10

On Off show-labels
On Off show-input
On Off show-functions
On Off show-matches

Figure 3. *A screen-shot of the simulator environment in NetLogo.*

Finally, the simulator includes a graphical interface through which experiment parameters can be set and values describing the state of the modelled hive can be monitored. Figure 3 shows a screen-shot of the implemented application. The animation of the hive is displayed on the window on the right side of the screen-shot, where the lines represent pheromone paths. On the left hand side we can use the controls to set values for various parameters affecting the course of the simulation. Indicatively, some of the parameters we have used are the pheromone evaporation rate, the probability of an outside ant carrying food to enter the nest, the probability of an ant to choose a particular pheromone trail, the number of entrances and initial trails etc. An extensive experimentation with the values of the above parameters will be able to provide valuable feedback on the ways they affect the life of the colony.

4. Discussion

In the process of modelling the behaviour of the Pharaoh's ants we have identified a number of issues on which an empirical comparison between the two methods is based.

There are advantages to both methods, though at different modelling levels. X-machines appear to be a natural metaphor for the modelling of the internal behaviour of ants because they can naturally describe their internal states and the transitions between these states caused by stimuli as well as represent the data structures which form the knowledge of an ant. However a Communicating X-machine model cannot by itself manage the required reconfiguration (in number of participating entities or communication channels), which is a prominent property of the system in question.

Population P Systems, on the other hand, possess a natural trait for capturing the self-assembly behaviour of the colony of ants and how the structure of the colony may change over time. However, when it comes to the modelling of an individual, PPS are less intuitive in representing their internal states and behaviour.

Consequently, it is clear that there is a valid rationale behind a possible combination of the two methods that uses a Communicating X-machine System to model the static structure of ants, their internal data and change of internal state and a Population P System with active cells to model the dynamic configuration of a colony by applying the reconfiguration operators on the Communicating X-machine System.

In terms of animation, we considered the case of NetLogo as a suitable platform to run certain experiments. The NetLogo implementation was carried out in a rather straightforward way, bearing in mind the formal XM models of the individual ants and the Population P System reconfiguration rules. The formal models have driven the creation of the simulation, by mapping the functions of X-machines and the re-write rules of P Systems to NetLogo functions. The implementation facilitated better understanding of the models and helped us correct some ambiguities that existed in the original models. The NetLogo implementation revealed a framework under which similar animations could be achieved in other domains of biology-inspired multi-agent systems. Work concerning the NetLogo animator involves automatic translation of XMDL specifications in NetLogo code. Some steps have already been taken towards this direction [Peneva & Kefalas, 2005]. Given the current structure of the X-machine meta-interpreter and the existing automatic XMDL to Prolog translation tools, this task is considered to be more than feasible.

5 Conclusions and Further Work

In this paper we use two different methods, namely Communicating X-machines and Population P Systems, in order to formally model Pharaoh's ant colonies. We have discussed the rationale and characteristics of each method and made a brief comparison between them in order to allow further development and improvement. A simulation of the ant colony based in NetLogo was also presented. NetLogo was suggested as a suitable tool for the development of prototype simulations and in-silico experimentation due to the friendly environment and tools it provides. The easiness in which experiments could be parameterised gave us the chance to make preliminary observations on fundamental issues of self-organisation and emergence that could be used to complement in-vitro experiments. Currently, we are investigating an integration of the two formal methods into a new one that will possess the prominent advantages of both. This attempt towards a kind of hybridisation of the two paradigms [Stamatopoulou et al, 2005b] gives rise to various issues which require further investigation, however we are confident that this new method can help us to model either other biologically-inspired artificial systems, such as autonomous nano-robotic swarms, or other complex systems, such as emergent complex web-services resulting from composition or aggregation of simpler ones.

References

Eilenberg, S. (1974). Automata, Languages and Machines. Academic Press.

Eleftherakis, G. (2003). Formal Verification of X-machine Models: Towards Formal Development of Computer-based Systems. PhD thesis, Department of Computer Science, University of Sheffield.

Eleftherakis, G. and Kefalas, P. (2000). Model checking safety critical systems specified as X-machines. Matematica-Informatica, Analele Universitatii Bucharest, 49(1):59– 70.

Holcombe, M. (1988). X-machines as a basis for dynamic system configuration. Software Engineering Journal, 3(2):69–76.

Holcombe, M. and Ipate, F. (1998). Correct Systems: Building a Business Process Solution. Springer-Verlag, London.

Ipate, F. and Holcombe, M. (1997). An integration testing method that is proved to find all faults. International Journal of Computer Mathematics, 63(3):159–178.

Kefalas, P., Eleftherakis, G., and Kehris, E. (2003a). Communicating X-machines: From theory to practice. In Manolopoulos, Y., Evripidou, S., and Kakas, A., editors, Advances in Informatics - Post-Proceedings of the 8th Panhellenic Conference in Informatics, volume 2563 of Lecture Notes in Computer Science, pages 316–335. Springer-Verlag, Berlin.

Kefalas, P., Eleftherakis, G., and Sotiriadou, A. (2003b). Developing tools for formal methods. In Proceedings of the 9th Panhellenic Conference in Informatics, pages 625–639.

Martin-Vide, C., Mauri, G., Paun, G., Rozenberg, G., and Salomaa, A., editors (2004). Membrane Computing: International Workshop, WMC 2003, Tarragona, Spain, July 17-22, 2003, Revised Papers, volume 2933 of Lecture Notes in Computer Science, Berlin. Springer-Verlag.

Paun, G., Rozenberg, G., Salomaa, A., and Zandron, C., editors (2002). Membrane Computing: International Workshop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19-23, 2002. Revised Papers, volume 2597 of Lecture Notes in Computer Science, Berlin. Springer-Verlag.

Peneva, K. and Kefalas, P. (2005). Animating formal models of biologically-inspired multi-agent systems. In Proceedings of the Balkan Conference in Informatics (BCI'05).

Paun, G. (2000). Computing with membranes. Journal of Computer and System Sciences, 61(1):108–143. Also circulated as a TUCS report since 1998.

Paun, G. (2002). Membrane Computing: An Introduction. Springer-Verlar, Berlin.

Stamatopoulou, I., Kefalas, P., Eleftherakis, G., and Gheorghe, M. (2005a). A modelling language and tool for Population P Systems. In Proceedings of the 10th Panhellenic Conference in Informatics, Volos, Greece, November 11-13.

Stamatopoulou, I., Kefalas, P., and Gheorghe, M. (2005b). Modelling the dynamic structure of biological state-based systems. BioSystems, 87(2-3): 142-149.

Wilensky, U. (1999). Netlogo. http://ccl.northwestern.edu/netlogo. Center for Connected Learning and Computer-based Modelling. Northwestern University, Evanston, IL.