ULB UNIVERSITÉ LIBRE DE BRUXELLES, UNIVERSITÉ D'EUROPE

Faculty of Science
Department of Computer Science

In partial fulfillment of the requirements for
the degree of Masters of Science in Computer Science 2011-2012

Formal Verification of Flexibility in Swarm Robotics

Lenaertz Mikaël

Supervisor
Prof. Mauro Birattari

Co-Supervisor
Manuele Brambilla

Abstract

In this master thesis, I propose two contributions to the study of flexibility in swarm robotics. First, I propose a novel definition of flexibility in swarm robotics. Second, I present an approach to the formal verification of flexibility in swarm robotics systems using model checking. A precise definition of flexibility in the context of swarm robotics is necessary, since limited effort has gone into formal definitions of the properties of swarm robotics systems. Even though flexibility is one of the most important properties of swarm robotics systems, together with robustness and scalability, the definitions available in the literature are usually not precise and sometimes flexibility and adaptability are mixed together. A precise definition of flexibility is however of little use in the development and analysis of swarm robotics systems. For this reason, I propose an approach to the formal verification of flexibility using model checking, and I validate this approach on a collective foraging scenario. Two different models are produced and compared with each other and with extensive simulated experiments. I also extend the result of model checking to predict possible behaviors of the system outside the tested parameters. Finally, I conclude with a discussion of the presented and future work.

Résumé

Dans ce mémoire, je propose deux contributions à l'étude de la flexibilité dans le domaine de la robotique en essaim. Premièrement, je fournis une nouvelle définition de la flexibilité dans le domaine de la robotique en essaim. Deuxièmement, je présente une approche de vérification formelle de la flexibilité des systèmes robotiques en essaim en utilisant la méthode du model checking. Étant donnés les efforts limités consacrés aux définitions formelles des propriétés des systèmes de robotique en essaim, une nouvelle définition précise de la flexibilité est nécessaire. Même si la flexibilité est l'une des propriétés, avec la robustesse et l'adaptabilité, les plus importantes des systèmes robotiques en essaim, les définitions disponibles dans la littérature ne sont généralement pas précises et, parfois, la flexibilité et l'adaptabilité se recoupent. Une définition précise de la flexibilité est cependant peu utile dans le développement et l'analyse de systèmes robotiques en essaim. Pour cette raison, je propose une approche de vérification formelle de la flexibilité en utilisant du model checking, et je valide cette approche sur un scénario d'exploitation de sources de nourriture. Deux modèles différents sont développés et comparés l'un avec l'autre en faisant de nombreuses expériences simulées. J'utilise également les modèles pour prédire les comportements possibles du système en dehors des paramètres testés en simulation. Enfin, je termine par une discussion sur le travail présenté et les travaux futurs possibles.

Acknowledgements

The last few years have been great. Hence, I would like to thank many people for their help in my master thesis and for making my university journey such a joy here at ULB.

First and foremost, this master thesis took a lot of time and effort not only on my part, but also on the part of my co-supervisor, Manuele BRAMBILLA. He was always there to help me resolve my problems, to answer my questions and to correct my master thesis. Therefore, I want to thank him. And finally, he made me discover *Turbosliders*, a game I played a lot with friends during breaks.

I would also like to specially thank my supervisor, Mauro Birattari. He believed in me from the start and was a great help and motivator.

I did my master thesis at the Iridia institute and would like to thank all the Iridia family for making me feel welcome. A special mention to Matteo Amaducci, Lorenzo Garattoni, Davide Vichi, Gaëtan Podevijn and Simon Lefort who made the Iridia student room the best place to be. I'm particularly grateful to Carlo Pinciroli and Arne Brutschy for their help and again to Carlo for the awesome LAN parties he organized. I am probably able to beat him now.

I would like to thank two great friends, Yassin Jdaoudi and Gabriel Corvalan Cornejo, for making the university years bearable, for their help, for their support and for the number of hilarious times we spent together, and Colas Goeminne, Paul Carette, Bruno Arsenne and Xavier Barthel for making my university life enjoyable.

A thank you goes also to all the previous and current members of the computer science association. Thanks to them, I learnt a lot of values, had great times and they contributed to making me the person I'm now.

There are much more people who influenced my university journey who I have not mentioned yet. I take opportunity to thank them all at once.

Finally, I would like to thank my family, especially my parents, Myriam Herman and Patrick Lenaertz, and my sister, Manon Lenaertz, for their support, their encouragements and for believing in me.

Contents

"When many work together for a goal, great things may be accomplished. It is said a lion cub was killed by a single colony of ants."

Saskya Pandita

Chapter 1

Introduction

Swarm Robotics focuses on how simple robots can cooperate in order to tackle complex tasks in a distributed way [7]. Up to now, a limited effort has gone into the direction of formally defining swarm robotics as an engineering field [8, 4]. In particular, swarm robotics is still characterized by fuzzy and conflicting definitions about its features and properties. The main goals of this thesis are: giving a new definition of *flexibility* in swarm robotics and examining a way to formally verify flexibility in swarm robotic systems.

Section 1.1 and Section 1.2 of this chapter define respectively swarm intelligence, swarm robotics and their properties. Section 1.3 gives the outline of this master thesis.

1.1 Swarm intelligence

In nature, there exists animal societies, consisting of large numbers of simple individuals, capable of accomplishing complex tasks. Typical examples are ant colonies, bird flocks, bacterial colonies and fish schools. Termite colonies, for example, composed from several hundreds to several millions individuals, can construct mounds up to 9 meters high (Figure 1.1a). *Swarming* or *swarm behaviors* are the terms usually employed to define such collective behaviors exhibited by animals.

The fascinating part of these animal swarms is that the individuals composing them are capable to coordinate their actions in a completely distributed way, despite changes in their environment and despite their lack of long range communication. A single individual would not be able to accomplish a specific tasks of the organism by himself. Only together they can accomplish the tasks. The perfect example of this is how weaver ants build their nests, which are made of leafs [12]. They construct it by weaving together leaves using larval silk. To do so, they must pull big leafs towards one another (Figure 1.1b). A single ant lack the strength to perform such a task by itself. However, a group of ants can gather around the leafs and start grabbing one leaf with their feet and the other one with their jaws. If the space between the two

(a) (b) (c)

Figure 1.1: Examples of swarm behaviors in nature. (a) a big mound constructed by a termite colony (from www.umfulana.com) (b) ants building a nest by pulling leafs together (from www.thephotosociety.org) (c) a flock of birds (from www.denmark.net).

leafs is to big, the ants interlock to form a chain between the leafs. The result of this distributed coordination is that the ant colony can build a complex nest, even if no single ant can move and weave two leaves together.

The above concept is called *self-organization*. Camazine et al. [11] defines it as "a process in which pattern at the global level of a system emerges solely from numerous interactions among the lower-level components of the system. Moreover, the rules specifying interactions among the system's components are executed using only local information, without reference to the global pattern". For instance, birds use only the position of their neighbor to move in a flock (Figure 1.1c).

Studies about swarm behaviors inspired researchers to use techniques abstracted from self-organized systems to solve practical problems (e.g, optimization and data analysis) [16]. The discipline that studies such behaviors is called *swarm intelligence*. Dorigo and Şahin [15] defines swarm intelligence as "the discipline that deals with natural and artificial systems composed of many individuals that coordinate using decentralized control and self-organization".

1.2 Swarm robotics

Swarm robotics is the application of swarm intelligence principles to multi-robot systems (Figure 1.2c) [15, 14]. The behavior of the robots in those systems are, in general, simple (Figure 1.2b). For this reason, a single robot is not able to solve a task alone. This is in contrast with the mainstream robotic research, where a complex robot needs to solve a task alone. One example is the iCub project, an ambitious project to construct a child humanoid robot (Figure 1.2a) [34].

Giving a good definition of swarm robotics is not easy. Şahin [1] proposed the following definition:

> Swarm robotics is the study of how large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among agents and between

(a) (b) (c)

Figure 1.2: Examples of existing robots (a) RobotCub, complex child humanoid robot (from www.technewsworld.com) (b) e-puck, relatively simple robot without complex behaviors (from www.e-puck.org) (c) a swarm robotic system of SwarmBots (from www.publications.csail.mit.edu)

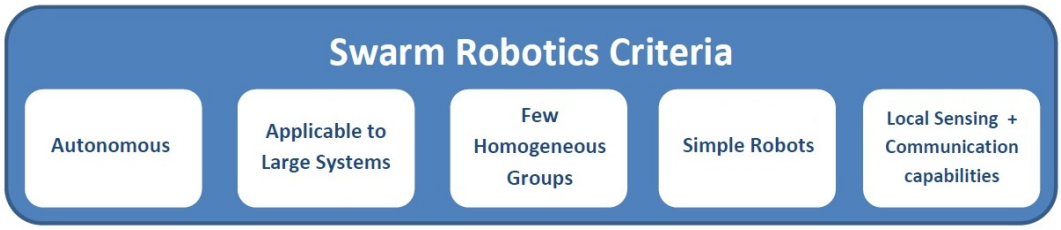


Figure 1.3: Set of criteria to define the degree to which a multi-robot system can be considered as swarm. (1) Autonomous: the system has no centralized controller and robots are able to interact with their environment (2) Applicable to large systems: the system should be scalable (3) Few homogeneous groups: the number of groups of the system should be small and the number of individuals in each group should be large (4) Simple robots: the robots have to be relatively incapable or inefficient (5) Local sensing and communication capabilities: coordination of the robots should be decentralized.

the agents and the environment.

Even though, this definition gives a good idea of what is swarm intelligence, it is not sufficient to differentiate swarm robotics from other multi-robot studies. In fact, some of the definitions of the different multi-robot fields overlap with each other. Therefore, Şahin [1] identified a set of criteria specific to swarm robotic (Figure 1.3).

The first criterion enunciates that the robots of a swarm system should be *autonomous*. The individuals composing a swarm robotic system are embodied and able to interact with their environment. This criteria exclude for example sensor networks, which consist of spatially distributed sensing elements. Such elements do not have physical interactions with the world. They only sense some difference in the data they observe and send new data to their neighbors, without acting.

The second criterion enunciates that swarm robotics systems should be designed to work with a large number of individuals. It's difficult to identify a lower bound number that differentiate swarm systems from non-swarm systems. In fact, in the literature, small groups of individuals are often considered as swarm systems. This is mainly due to the fact that it is not always feasible to execute experiments with

large number of robots. Nevertheless, swarm robotics studies done with small swarm systems should be executed with *scalability* in sight.

The third criterion enunciates that swarm systems should contain *few homogeneous groups* of robots. The number of groups in a swarm robotic system should be small and the number of individuals in each group should be large. A homogeneous group of robots consists of robots which are physically identical and have the same behavior. An example of robot systems not respecting the third criteria are robosoccer teams [37]. Each robot of a team has a different role depending on his position on the football field: goalkeeper, defender, midfielder, attacker. The reason why it does not respect the criteria is that there are only a few robots in the four groups. A positive example is the Swarmanoid project in which there are three different robot groups consisting of tens of robots each (foot-bots, hand-bots, eye-bots) [17].

The fourth criterion enunciates that swarm system robots should be *simple*. In swarm systems, robots are relatively incapable or inefficient on their own. That is, either 1) cooperation among them should be essential to tackle a task, or 2) adding more individuals to a swarm should improve the performance and robustness of the system. However, the criterion is not a restriction on the hardware and software of the robots. It simple states that it is possible to use simple robots instead of complex ones to tackle a task.

The last criterion is that robots of a swarm system should have *local sensing* and *communication capabilities*. Consequently, swarm systems should be *decentralized*. As the number of robots in a swarm system becomes larger, the number of interactions between them and a central station becomes larger as well. This can create bottleneck phenomena that can severely reduce the scalability of the system.

These criteria promote interesting properties in swarm robotic systems:

- *Robustness*: "the ability of a swarm robotic system to continue operating, although at a lower performance, despite disturbances in the environment or failures in individuals" [1]. For example, in nature, ants continue foraging even if half of them have been killed.
- *Scalability*: A swarm robotic system is scalable if it is able to operate under a large range of group sizes. For instance, the flocking behavior of birds still display similar behaviors if it is composed of 10, 100 or 1000 individuals.
- *Flexibility*: Since flexibility is the main topic of this thesis, I will delay its definition and discuss flexibility in detail in the next chapter.

1.3 Thesis outline

This master thesis is organized as follows:

Chapter 2 propose a new definition of flexibility in swarm robotic systems and

propose a way to formally verify it.

Chapter 3 presents the background of this work. This chapter begins by introducing the state of the art of model checking in swarm robotics. Afterwards, it presents the tools needed for the experiment.

Chapter 4 introduces in the first section the swarm robotic experiment used for formal verification of flexibility. The second section describes how to implement the experiment in the ARGoS simulator. The third section describes two different approaches to model the chosen experiment. Those models are used for formally verifying flexibility of the experiment simulations. The chapter ends with a comparison of the results obtained using a the ARGoS simulator and using the models. In the last section, a conclusion is also given about the formal verification of flexibility in swarm robotic systems.

Chapter 5 summarizes the results and contributions of this master thesis. This chapter ends by giving an outline of the possible future work that can be done about flexibility in swarm robotics.

Chapter 2

Flexibility

The aim of this work is to propose a new definition of flexibility in swarm robotic systems and to propose a way to formally verify it. Flexibility has, in the literature, various definitions, some of which are given bellow. The problem of the current definitions of flexibility is that they are very often fuzzy or contradictory. Moreover, there is no clear distinction between flexibility and adaptability.

2.1 Definition

The Cambridge Dictionary defines flexible as "able to change or be changed easily according to the situation" and defines adaptable as "able or willing to change in order to suit different conditions" [29]. These two definitions overlap with each other and the difference is not clear. However, it is possible to define them, in the swarm robotic field, so that there is a clear difference between them.

In this master thesis, I define flexibility as:

> The ability of a system to achieve similarly performing collective behaviors in diverse environments without changing the behaviors of the individuals of the system.

and I define adaptability as:

> The ability of a system to generate solutions to different tasks by using different coordination strategies. The behavior of the individuals adapt to the requirements of the tasks.

In this thesis, I only focus on flexibility, leaving adaptability for a future work. In my definition, I do not define clearly what means for a system to 'achieve similarly performing collective behaviors'. This is done on purpose, as this part of the definition strongly depends on the application and on the requirements of the system

Foraging ants are an example of a swarm that is flexible and a robotic system that needs a map to forage is an example of a swarm that is not flexible. When putting ants in another environment, they still be able to forage. That will not be the case of the robotic systems, they need a map of the new environment to forage. However, by mimicking the behavior of the ants with robots, the robotic system can forage in different environments without the need for maps.

I present some definition of flexibility that can be found in the literature so that I can discuss their limits and compare them with my definition of flexibility:

Şahin [1]: "the ability of a swarm robotic system to generate modularized solutions to different tasks". Şahin [1] also states that "swarm robotic system should have the flexibility to offer solutions to the tasks at hand by utilizing different coordination strategies in response to the changes in the environment". This definition of flexibility seems more appropriate for adaptability.

Blum and Merkle [6]: "The individuals of a swarm should be able to coordinate their behaviors to tackle tasks of different nature". Like the definition of Şahin [1], it seems also more appropriate for adaptability.

Bayindir and Sahin [3]: "The capability to adapt to new, different, or changing requirements of the environment". This definition seems also more appropriate for adaptability.

Lerman et al. [30]: "The ability to reallocate and redistribute units in a self-organised way when units are dynamically added or removed". This definition of flexibility overlaps with the definition of robustness and scalability.

Garnier et al. [21]: " The ability for a system to readily adapt to new, different, or changing requirements. Flexibility of self-organized systems is well illustrated by the ability of social insects to adapt their collective behaviors to changing environments and to various colony sizes. These adaptations can occur without any change of the behavioral rules at the individual level". This definition is similar to our definition. The problem is that it uses the word "adapt". Consequently, there is no clear difference between flexibility and adaptability.

2.2 Verification

In literature, flexibility is usually tested in an informal way. A behavior is run a limited number of trials on a set of different environments. These environments differ according to a set of parameters that describe the environment. A parameter can be the size and form of the environment, the weight and distribution of objects, the duration of certain events,... For example, Kube and Bonabeau [26] tested his cooperative transport behavior on various environments and verified if the behavior worked on them. His experiment consisted of robots pushing objects towards a goal. The different environment parameters he changed were the box types (bigger,

different forms), the goal position,... The goal of verification of flexibility is to check in which ranges of the environment's parameters the behavior works properly. This approach is not very systematic. In order to give a better verification of flexibility a more formal and scientific approach is necessary

In our method, the formal verification of flexibility is based on model checking [13]. It is done by following 4 steps. I suppose that the system to analyze is already available and cannot be modified.

1. Develop a model of the system. In particular, the model to develop must be a *Continuous-Time* or *Discrete-Time Markov Chain* [25].

2. Define the metric used to evaluate the performance of the system. In particular, the metric must be specified using a *Probabilistic Computation Tree Logic Formula* [22].

3. Validate the model against the system. It is necessary to verify whether the developed model produces results which are comparable to those obtained by the system. To do this it is necessary to choose a set of test cases and compare the results.

4. Once the model is validated we can use it to verify that the performance of the system are acceptable over a range of parameters.

I validate the proposed way by means of an example in foraging. To do so, I create a flexible swarm behavior able to forage in different environments. More details about the experiment are given in Chapter 4.

Chapter 3

Background

As described in Chapter 2, in this master thesis, flexibility is verified by using model checking. This chapter introduces in Section 3.1 model checking and other formal verification techniques and presents in Section 3.2 the tools that were used for the case study of this master thesis.

3.1 Related work

In this section, I provide the knowledge required for understanding formal verification and model checking. Section 3.1.1 gives an insight of formal verification and the existing techniques. Section 3.1.2 provides the principles of model checking. Section 3.1.3 lays out how to use model checking in swarm robotic systems. Section 3.1.4 illustrates the theory of the previous sections with a practical example. Finally, Section 3.1.5 introduces property-driven design, which is a new approach to design and develop swarm robotic systems.

3.1.1 Formal verification

When developing a swarm robotic systems, the developers have a specific desired collective behavior in mind, but they cannot directly program it. In fact the collective behavior is the result of the interaction of the individual behaviors, which is, in general, unpredictable [24]. Therefore, the development of such systems is difficult. Swarm robotic systems are usually developed through a trial and error process, until the desired result is reached [8]: the developer first designs the individual behaviors of the swarm system and then check the resulting global behavior on real or simulated robots. If the resulting behavior is not the one desired, the individual behaviors are modified and the process is repeated until the desired collective behavior is obtained. This method is not effective because it is time consuming and does not give any assurance that the wanted result is obtained.

Furthermore, checking all the possible outcomes of the interactions of robots in

simulation or on the real robots is often impossible, as the number of possible executions is very large in most systems [28]. Fortunately, there exist methods that can check all the possible outcomes of the executions of a system and thus validate it. These methods, known as deductive verification and model checking, are introduced in the following paragraphs. Using deductive verification and model checking it is possible to check every possible execution of a system and find unexpected behaviors [13]. This can be crucial in applications where failures are unacceptable. For example, in 1996, the explosion of the Ariane 5 rocket forty seconds after it was launched, was due to an error that occurred during the conversion of a number (from 64-bit to 16-bit) [18]. This kind of severe errors could also occur in swarm robotics systems, for instance in rescue missions. Critical errors are very difficult to find with standard testing techniques but could be found with formal verification techniques.

In literature, *formal verification* is defined as the process of checking whether a system, hardware or software, satisfies some requirements [2]. More attention has been given in research for hardware than for software verification [23]. The reason for this is that an error in hardware systems has a bigger impact than a software error and it is more difficult to fix a posteriori. However, software and hardware verification techniques share the same basic ideas. The principal formal verification techniques are [13]:

Simulation and testing

The simplest formal verification techniques are *simulation* and *testing*. Those techniques imitate the interactions of a system in order to verify if the system works as intended. However, they do not cover all the possible outcomes of a system. Those techniques use simple input-output tests to check specific cases of the system. The difference between them is that simulation is performed on a model of the system, whereas testing is performed on the real system. The advantage is that those techniques are a cost-efficient way to find many errors and that they are close to the system, indeed they do not abstract away from the details of a system. Simple input-output tests can be done easily on all the parts of the code and can discover a lot of errors, but might not be enough for safe critical systems.

Deductive verification

Deductive verification is a formal verification technique allowing to derive (or deduct) conclusions from one or more general statements (premises, axioms) which are simple truths about a system. An example of deductive reasoning is [38]:

1. All men are mortal.
2. Socrates is a man.
3. Therefore, Socrates is mortal.

The first statement enunciates that all objects labeled as “man” are considered as mortal. The second statement stipulates that the object “Socrates” is labeled as a “man”. Consequently, we can deduct that “Socrates” is mortal because he inherits this property from the fact that he is labeled as a “man”.

This kind of verification was, in early research, used for guaranteeing the correctness of critical systems. The importance of the correctness of software and hardware systems was sometimes so great that mathematicians or logicians spent lots of energy and time to proof by hand the correctness of the systems. Nowadays, there exist tools, such as the prolog language and environments [9], that simplify this processes of deductive verification. However, deductive verification is still a time-consuming process that require expertise about logical reasoning. For these reasons, deductive reasoning is only used in projects were having no failures is essential. For instance, projects like space or rescue missions. Given its relative simplicity and speed, model checking is the optimal candidate to perform formal verification of properties on swarm robotics systems. For this reason, I use model checking in this thesis.

Model checking

Model Checking is a formal verification technique that is able to verify properties of finite state concurrent machines. More information about state machines are given in section 3.1.2. The specifications of a system are verified by performing an exhaustive search on the state space. This allows to determine if the specification is satisfied. Although model checking is restricted to finite state machines, it is still applicable to infinite state machines. This is an interesting property because most of swarm robotic systems are infinite state concurrent machines.

3.1.2 Principles of model checking

In this section, I give the knowledge needed for constructing a model of a system. I present the basic knowledge of *state machines, Probabilistic Temporal Logic (PTL), Computation Tree Logic (CTL)* and the advantages of *statistical model checking.*

Finite state machines

A *finite state machine* is a mathematical model of computation used for describing hardware and software systems by the mean of an abstract machine containing a finite number of states and a finite number of transitions between the states [5]. A state represents a *configuration* of the system at a given time and a transition represents the passage or link of the system from one state to another. A state machine can only be in one state at a time, called the *current state.* It changes his current state to another state if there exist a transition between the two and if the condition of the transition is satisfied. Moreover, a transition’s condition can be deterministic or probabilistic. A *Markov Chain* (MC) is a finite state machine where

the condition of the transitions are probabilities. Several kinds of state machines are available, but only two are of interest for this work: *Discrete-Time Markov Chains* (DTMCs) and *Continuous-Time Markov Chains* (CTMCs) [28].

Definition 1. *A DTMC is defined as a tuple* $(S, \bar{s}, P)$ *where*

- S *is a finite set of states;*
- $\bar{s}$ *is the initial state;*
- $P : S \times S \to [0; 1]$ *is the transition probability matrix where* $\sum_{s' \in S} P(s, s') = 1$ *for all* $s \in S$*.*

$P(s, s')$ *represent the probability of going to state* s' *when the current state is in* s*.*

DTMC assumes discrete time, that is, time is divided in *timesteps* and in each timestep only one event can happen. CTMC instead considers continuous time.

Definition 2. *A CTMC is defined as a tuple* $(S, \bar{s}, R)$ *where*

- S *is a finite set of states;*
- $\bar{s}$ *is the initial state;*
- $R : S \times S \to \mathbb{R}_{\geq 0}$ *is the transition rate matrix.*

R *assigns a rate to each transition.*

When a *rate* $R(s, s')$ is bigger than 0, then the probability that the transition from s to s' is triggered within t time-units equals

$$1 - e^{-R(s,s') * t}$$

When $R(s, s')$ equals 0 there is no transition from s to s'. One of the characteristics of rates is that the probability to be triggered after an amount of time t at a moment t_i or at another moment t_j is the same.

In CTMCs, the non-zero transition rates of a state are in *race conditions*, in other words, the first transition triggered is the one that determines the next state.

Paths in DTMCs and CTMCs

A single execution of a system can be represented using a DTMC or CTMC by a sequence of states, called a *path*. The path is infinite if no final state is present. However, in swarm robotics, experiments are usually stopped after an arbitrary amount of time which transform them in finite Markov chains. In DTMC, the execution time of the experiments defines the number of timesteps evaluated and

thus the length of the path. The length of a path is equal to the time limit divided by the duration of a timestep.

In CTMC, the number of states in a path depends on the time that the machine stays in each state, which is not fixed. Therefore, the length of a path can vary enormously in CTMCs. The time spent in a state $E(s)$ (*exit rate*) of a CTMC is calculated by using the addition of the rate of all the outgoing transitions:

$$E(s) = \sum_{s' \in S} R(s, s')$$

As seen in the previous section, with a rate it is possible to calculate the probability to stay t time-units in a state. Consequently, we can compute the average time that a machine will stay in each state.

Definition 3. *A path w in DTMCs or CTMCs is a sequence of states $s_0 s_1 s_2 ... s_n$ such that:*

- *s_0 is the initial state.*
- *$s_i \in S$.*
- *n is the length of the path $|w|$.*
- *$P(s_i, s_{i+1}) > 0$ for DTMC (or $R(s_i, s_{i+1}) > 0$ for CTMC) for all $i \in [0, n-1]$.*

All the possible execution of a system can be seen as a finite number of paths. Therefore, it is possible to calculate the probability of a path to be taken, that is, the probability of observing a particular run of the system. In DTMCs, the probability measure of a path is defined as the product of the transition probabilities. In CTMCs, the probability of execution of a path can also be calculated through a simple transformation of the outgoing transition rates into probabilities. It is done by dividing the rate of a outgoing transition of s by $E(s)$, which is defined above as the sum of the outgoing transition rates of s. However, finding all the possible paths in CTMCs can be done for a defined number of states but not for a limited amount of time. This is because, as previously seen, the amount of time stayed in a state is not fixed. To calculate all the possible paths for a fixed amount of time in CTMCs, the time stayed in each state must be added in the calculation. The computation of the probability of a path is necessary when verifying properties of the system. For example, computing the percentage of the paths respecting a property.

Probabilistic computation tree logic

Properties of DTMCs and CTMCs can be expressed using a *temporal logic* [28]. The logic that is most often used with DTMCs in swarm robotic literature is *probabilistic computation tree logic* (PCTL), which is a probabilistic extension of the temporal logic *Computation Tree Logic* (CTL) [22]. CTL is a branching time logic based on the idea that a Markov chain (MC) can be expanded in a computation tree

where the *root* is the initial state of the MC and where each node is a possible state of the system [8]. PCTL add the *probabilistic operator* $P_{\sim r}[\varphi]$ where r is a bounded probability satisfying $0 \leq r \leq 1$, φ is a *path formula* and $\sim \in \{\leq, \geq, <, >, =\}$. A state satisfies $P_{\sim r}[\varphi]$ if the probability of taking a path, from s, satisfying φ in the interval defined by $\sim r$. The different path formulas for a state are:

- $\bigcirc\phi$ is true if ϕ is satisfied in the next state.
- $\Diamond\phi$ is true if ϕ is satisfied in the current state or in one of the next states.
- $\Box\phi$ is true if ϕ is satisfied in the current state and all the next states.
- $\phi U \psi$ is true if ϕ holds until ψ holds.
- $\phi U^{\leq k} \psi$ where $k \in \mathbb{N}$ is true if ψ is satisfied in k steps and ϕ is satisfied until then.

where ϕ and ψ represent state formulas.

PCTL also define quantitative measures properties that uses cost and reward structures: real values associated with certain states and transitions. The global cumulative cost and reward can be calculated by summing the costs and rewards encountered during a run. The quantitative operator R works in a similar way as the probabilistic operator P:

- $R_{\sim r}[C \leq k]$ is true if, after k timesteps, the expected accumulated reward satisfies $\sim r$.
- $R_{\sim r}[I = k]$ is true if, at timestep k, the expected state reward satisfies $\sim r$.
- $R_{\sim r}[F\phi]$ is true if, before reaching a state satisfying ϕ, the expected accumulated reward satisfies $\sim r$.

where $r \in \mathbb{R}_{\geq 0}$, ϕ is a state formula, $k \in \mathbb{N}$ and $\sim \in \{\leq, \geq, <, >, =\}$.

For CTMCs, the probabilistic temporal logic that is often used is *Continuous Stochastic Logic* (CSL), which is, as PCTL, also an extension of CTL. The difference with PCTL is that in the path formulas the k of the until operator $U^{\leq k}$ is replaced with a non-negative real number, which represent a time instance in place of a timestep.

Verification

When verifying properties of a system, the developer can decide to use probabilistic or statistical model checking [25]. The difference is that statistical model checking only verify the properties on a sample of the possible outcomes [20], whereas probabilistic model checking explore all the possible situations. However, the problem of probabilistic model checking is that it can be time consuming and it needs

a lot of memory. It also requires the building of the complete DTMC or CTMC model, which is not the case of statistical model checking.

Statistical model checking explores a limited number of runs extracted from the model. In order to decide if a sample of runs is significant enough, the reliability of the estimator, which is obtained by computing the mean of the sample, is calculated. This reliability can be characterized by:

- Confidence interval (CI): is an interval of width $2w$ such that, if several samples of the same size are randomly obtained, the real probability is in those intervals $100 \times (1 - \alpha)\%$ of the times, where α is the level of confidence.
- Asymptotic confidence interval (ACI): which is basically the same as CI, but the variance is estimated using the Normal distribution. This method is appropriate when the number of samples is large.
- Approximate Verification of Discrete and Continuous Time Markov Chains (APMC): which approximate the real probability by running a certain number of generations.

In some system, certain properties can be critical and must be satisfied in all the possible situations. Statistical model checking cannot guarantee that and therefore probabilistic model checking must be done. In swarm robotics, the size of the models is usually big, thus probabilistic model checking would be time taking and space demanding. Indeed, probabilistic model checking needs that the entire model is constructed. This is not the case for statistical model checking. While it is not being complete, it gives a formal guarantee of the properties to verify under certain known limits. This makes statistical model checking substantially different then simulation and testing.

3.1.3 Model checking in swarm robotics

The approach used in this work to formally verify flexibility in swarm robotic systems is based on the work done by Konur et al. [25]. They use probabilistic model checking to verify properties of a swarm robotic system. The model checker that is used in this work is PRISM, more information about it is given in Section 3.2.4.

This section lays out the micro- and macroscopic approach for modeling a swarm robotic system, some formulas allowing to transform probabilities, and time distributions to rates, and an example of a DTMC and a CTMC model of a swarm robotic system.

Micro- and macroscopic approach

There are two different approaches for modeling a swarm robotic system using a finite state machine: the microscopic approach and the macroscopic approach [25].

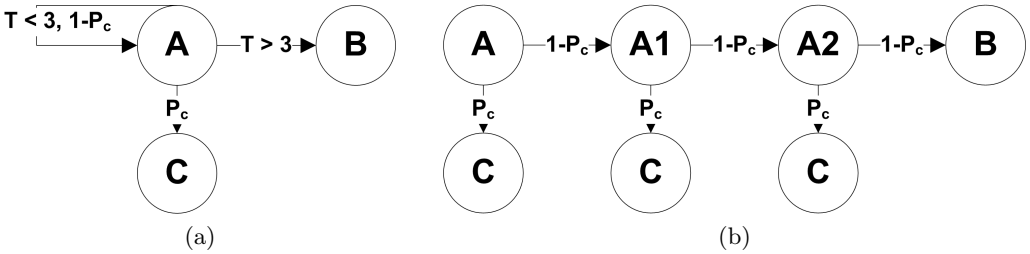


Figure 3.1: The transformation of a finite state machine with a transition containing a time condition (a) into a markov chain (b), which is a finite state machine with only probabilistic conditions.

Number of robots		1	2	3	4	5	6
Microscopic	**DTMC**	10^2	10^4	10^6	10^8	10^{10}	10^{12}
	CTMC	10	10^1	10^2	10^3	10^4	10^5
Macroscopic	**DTMC**	10^2	$\sim 5 \times 10^3$	$\sim 1.7 \times 10^5$	$\sim 4.4 \times 10^6$	$\sim 9.1 \times 10^7$	$\sim 1.6 \times 10^9$
	CTMC	10	55	$\sim 2.2 \times 10^2$	$\sim 7.1 \times 10^2$	$\sim 2 \times 10^3$	$\sim 5 \times 10^3$

Table 3.1: Comparison of the state space growth between a DTMC and a CTMC model with the micro- and macroscopic approach. The example considers that the robot's behavior contains 10 states with each a maximum time condition of 10 timesteps. In the example, an assumption of no limit on the number of robots in a same state is done.

The first approach consists of building a model by first creating a corresponding finite state machine for each robot behavior and then by taking the composition of all those finite state machines. The second approach consists of building a single finite state machine containing a state for each different sub-behavior of the robots. Each state is associated with a counter that keep track of the number of robot currently in that state.

The microscopic approach is really useful for analyzing in detail the interactions between robots but the state space grows exponentially with the number of robots. In fact, Table 3.1 shows that a state machine of a swarm behavior with 100 states has a state space composed by 10^{12} states with only 6 robots. PRISM, the state of the art model checker used in this work, can handle up to 10^{10} with probabilistic model checking, making a microscopic analysis with more than 5 robots computationally prohibitive.

100 states for describing a behavior may seem enormous, but in DTMCs, transitions with time conditions are replaced by a number of states equal to the number of steps defined by the time condition (see Figure 3.1). An instance of a behavior containing 100 states would be one with 10 states having each a time condition of 10 timesteps.

A method to limit the fast state space grow problem, is the macroscopic approach. The macroscopic approach allows us to overcome the problem of large state

spaces in DTMCs and CTMCs by using a population model, i.e. a model that only contains a state with a counter for each sub-behavior of each type of robot.

Table 3.1 shows that the state space size of a DTMC with the macroscopic approach grows at a slower rate than with the microscopic approach. The difference with only 6 robots is a model composed of almost 10^3 times fewer states, which is already a good improvement.

The formula to compute the number of states of a state machine with the macroscopic approach is [19]:

$$\binom{N+K-1}{N}$$

where K is the number of states and N the number of robots.

Constructing a CTMC model with the macroscopic approach of a system will, like with the microscopic approach, reduce the number of state due to the fact that the time conditions are being implied in the rates. However, if there are no time conditions, constructing a model by using a DTMC or a CTMC will give the same number states.

Probabilities, times and rates

Swarm robotic systems usually are characterized by two types of transitions: time conditions and probabilities. In DTMCs, time conditions substituted by adding a certain number of states and probabilities are the transitions conditions. However, CTMCs cannot directly express those two types of transitions. Therefore, if we want to build a CTMC of a swarm robotic system, every time condition and probability must be transformed into corresponding rates. The transformation formulas are defined as follow:

Definition 4. *Probability to rate:*

$$r = \frac{-ln(1–P)}{t}$$

where P is the probability over a time period t in seconds.

Definition 5. *Average time to rate:*

$$r = 1 - e^{-1/t}$$

where t is the average time of the time condition in seconds.

In CTMC models with the macroscopic approach, the transitions depend on the number of robots in the state. Thus, the rates are multiplied by the number of robots in the corresponding state and by consequence the rate becomes higher.

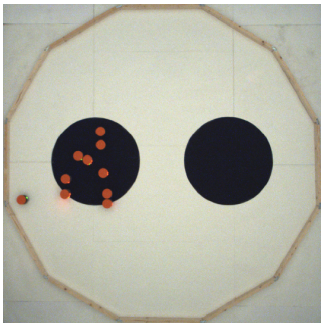


Figure 3.2: The environment of the aggregation application with 10 robots and 2 aggregation areas (from [8]).

3.1.4 Preliminary experiments

In this section, I present a DTMC and a CTMC model of a swarm robotic system to illustrate the concepts defined above and perform a comparison between them. It is important to remark that the values I use for the parameters of the models are just chosen to illustrate an example and might not correspond to those of a real system. The focus of this section is on the creation and the performance of the models and not on fitting the parameters.

The swarm robotic system used is the aggregation collective behavior described by Brambilla et al. [8] in their paper "Property-driven design for swarm robotics". The aggregation collective behavior is defined as follow:

> The robots have to cluster in one of the two aggregation areas of the environment (see Figure 3.2). The difficulty is that they do not have neither information about the environment nor information about the exact position of the other robots.

The robots go around at random and stop if they encounter a black spot on the floor that represent an aggregation area. According to a certain probability, they leave the aggregation area and restart walking randomly in the environment. This behavior of the robots can be modeled with a DTMC containing three states (see Figure 3.3): state A, a robot is in the first aggregation area, state B, a robot is in the second aggregation area, and state C, a robot is in the remaining part of the environment. A robot can go from state C to A, C to B, A to C, B to C or stay in the same state. The different probabilities of the transitions are set as follow:

- $P_{ca} = P_{cb} = \frac{S_{agg}}{S_{all}}$, where S_{agg} is the surface of one aggregation area and S_{all} is the surface of all the environment.
- $P_{aa} = 1 - P_{ac}$, $P_{bb} = 1 - P_{bc}$ and $P_{cc} = 1 - P_{ca} - P_{cb}$.
- $P_{ac} = P_{bc} = P_{max} \times (1 - \frac{N_s}{N})$, where N_s is the number of other robots sensed by a single robot, N is the total number of robots in the environment and P_{max}

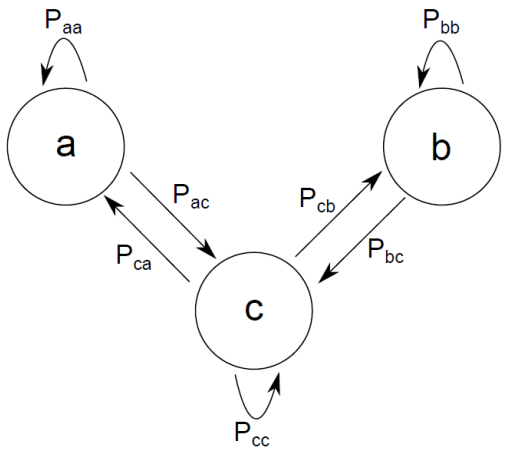


Figure 3.3: DTMC of a robot's behavior of the aggregation application. There are tree states (A, B, C) and seven transitions with probability conditions(P) (from [8]).

is the maximum leaving probability.

To create a DTMC model of the swarm robotic system with the microscopic approach, we simply have to take the product of the behavior (DTMC model) of each robot. In this case, we take n times the DTMC showed in Figure 3.3. For the macroscopic approach instead, only the DTMC model of one robot is needed. A counter, going from 0 to the number of robots in the system, is associated to each state of the DTMC model. For simplicity, we assume that just one robot can change state at each timestep. The probabilities of the transitions are calculated as follow:

- P_{ca}, P_{cb}, P_{cb} and P_{ca} are defined with the same values as the ones in the DTMC model of a single robot.
- $P_{nothing} = 1 - P_{ac} - P_{bc} - P_{cb} - P_{ca}$, is the probability that no robot changes state.

The CTMC model of the behavior of one robot resembles to the DTMC one. The difference is that all the probabilities are replaced by rates (see Figure 3.4). For example, the rate of going from state C to state A equals $\frac{-ln(1-P_{ca})}{t}$, where t is the length of a timestep in the DTMC model and P_{ca} is the probability of one robot to go from state C to A. In the same way as with the DTMC model of the aggregation system with the macroscopic approach, a counter is added in each state. The rates of the transitions are multiplied by the number of robots in the state. Similar to the DTMC model, we assume that only one robot can change his state during a timestep.

Properties of those aggregation systems could be, for example, the number of robots does not change over time $P_{\geq 1}\Box(N = N_{init})$, where N is the number of robots and N_{init} is the number of initial robots, and the probability to have an aggregation in one of the two areas in less 1000 seconds/timesteps (depending if we deal with

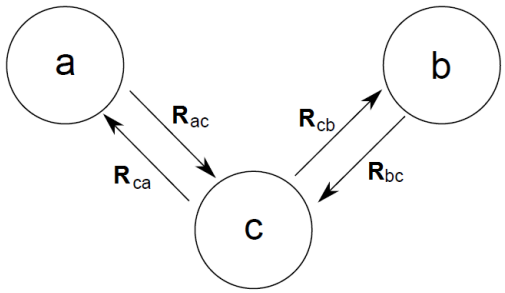


Figure 3.4: CTMC of a robot's behavior of the aggregation application. There are tree states (a,b,c) and seven transitions with rates (R).

N	Model type	Num. of states	Build T. (s)	Verif. T. (s)	P1 (%)
10	DTMC	66	0.021	0.006	91.3
	CTMC	66	0.01	0.005	91.6
50	DTMC	14504	0.222	0.181	4.6
	CTMC	14504	0.06	0.074	5.2
100	DTMC	31016	0.527	1.101	0
	CTMC	31016	0.146	0.383	0
500	DTMC	125751	15.332	78.22	0
	CTMC	125751	3.777	19.639	0

Table 3.2: Comparison between the DTMC model and CTMC model in term of performances. N represent the number of robots, Buid T. is the construction time of the model, Verif T. is the verification time of P1 with probabilistic model checking and P1 is the probability to have an aggregation in one of the two areas in less 1000 seconds. The tests were run on a HP Pavilion dv6, hosting Ubuntu 11.04, with a Intel(R) Core(TM) i3 CPU 4GB RAM @ 2.40GHz.

CTMCs or DTMCs) $P_{=?}[F \leq 1000\ (a = N_{total})|(b = N_{total})]$, where N_{total} is the number of robots and F represent the spent time.

We can see that it is quite simple to describe a system using a markov chain. The question that emerges after constructing the two models is "Which one performs better?". In the previous section, it was obvious that the macroscopic approach is better in terms of performance than the microscopic approach, but no conclusion could be done about the difference in performance between the DTMC model and CTMC model of a system without time conditions. Therefore, I ran some experiments on the two models (see Table 3.2). The PRISM code of the two models can be seen in section A.1.1 and A.1.2 of the appendix.

Table 3.2 shows us that the number of states of the two models stay the same and the results are similar. However, the construction and verification time of the CTMC model are faster than the construction and the verification time of the DTMC model. That makes the CTMC a better choice for the experiments of the next chapter. Additionally, continuous time is the most trivial manner to describe behaviors

of robots in time due to the fact that the real world is in continuous time also. Nonetheless, DTMC still have sense due to the fact that real or simulated robots often work in time cycles, but that is a problem that developers can overcome easily for the CTMCs. The result of this comparison is that the difference between DTMC and CTMC is little. In this thesis I decided to use a CTMC approach.

3.1.5 Property-driven design

I stated previously that, usually, swarm robotic systems are developed using a code-and-fix approach. In fact, the individual behavior of the robots are implemented, tested and modified until the desired collective behavior is obtained. In order to augment effectiveness, Brambilla et al. [8] proposes to develop swarm robotic systems by using a top-down design method, called property-driven design. It consist of a sequence of steps defined as follows (Figure 3.5):

Phase One consists of formally defining the requirements of the system by stating the desired properties.

Phase Two consists of building a model that satisfies the properties stated in the first phase. It is an iterative process: the developer must construct the model by small steps and improve it until all the properties are satisfied. The obtained model after this phase is robot independent. The developer can now identify the capacities needed by the robots in order to perform the required actions of the system and, consequently, choose the appropriate robot type.

Phase Three consists of implementing a simulation of the swarm robotic system described by the model in phase two. It is possible that the simulation does not validate the model, if so, the developer must go back to phase two and modify the model so that it includes the new information of phase 3 and still satisfies the requirements of phase 1.

Phase Four consists of implementing the system on real robots. Similarly as in phase three, if the implementation on the real robots does not validate the simulation, the developer must go back to phase three and modify the simulation so that it obtains the same results in phase four and still respect the model of phase two.

The point of property-driven design is to guide developers in their implementation of a swarm robotic system. It reduces the risk of not satisfying requirements of the wanted system, by checking the validation of the properties at each phase. The big difficulty in property-driven design is phase two, since it relies on the skills of the developer to model the global behavior of a system satisfying the properties of phase one. A last advantage of the property-driven design is that it makes it possible to decide the hardware after having a first insight of the system.

This approach is not used in this master thesis because I preferred to focus on the verification of flexibility in swarm robotic systems. Nevertheless, it is presented

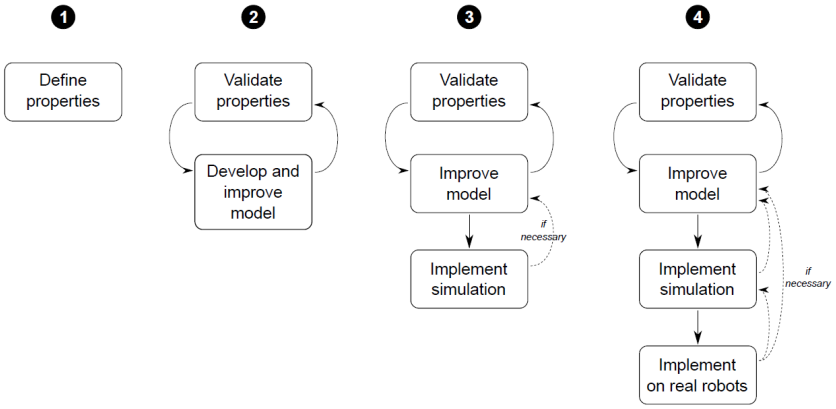


Figure 3.5: The four steps of the property-driven design for developing swarm robotic systems. (from [8])

here to clarify the complete context of this thesis. In the future, we plan to introduce a formal study of flexibility and other general properties as a necessary step in the design process of swarm robotics systems.

3.2 Tools

In order to formally verify flexibility in a swarm system, we used various tools: a simulator able to handle large swarm systems, small robots rich in functionalities and a model checker. The environments of the experiment explained in Chapter 4 also require IRIDIA TAMs, which are "booths" in which robots can enter.

In this section, the chosen tools for the experiment are introduced. In Section 3.2.1, the ARGoS simulator is presented. In Section 3.2.2, the e-puck robot is described. In Section 3.2.3, a presentation of the IRIDIA TAMs is given. In Section 3.2.4, the model checker PRISM is presented.

3.2.1 The ARGoS Simulator

When studying swarm robotics systems, researchers need to undertake experiments to analyze the swarm behaviors they conceived. Doing those experiments on real robots is sometimes impossible. In particular, to test properties like scalability, it would be necessary to use hundreds of robots, which is very often unfeasible. The cost of such large numbers of robots would be very high. Furthermore, simply test or debug such systems would be really time consuming. In fact, the behavior must be installed on each robot every time the developer wants to test his new implementation. Another drawback is the risk that a robot damages itself as result of an unexpected behavior. Consequently, researchers often use simulators.

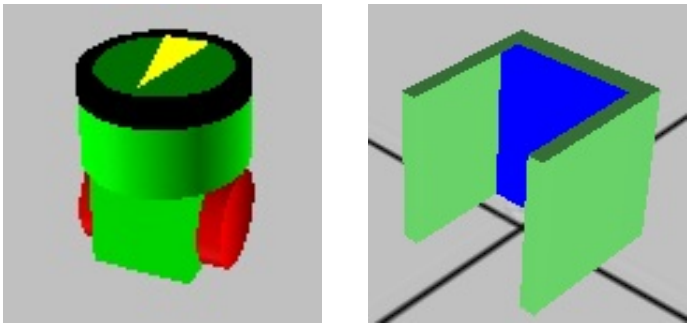
A good swarm robotic simulator should focus on being able to run with high number of robots. In this master thesis we use the ARGoS simulator. *ARGoS*

(Autonomous Robots Go Swarming) is an open source robot simulator developed within the *Swarmanoid Project* [17] by Carlo Pinciroli and his colleagues from the IRIDIA-ULB laboratory [36]. It is thought esplicitly for swarm robotics and thus it is able to run experiments composed of thousands of robots.

Figure 3.6: The ARGoS simulator running a collective foraging system in a arena delimited by red walls. The gray floor represent the nest, the yellow spots represent lights, the red spots represent prey that the robots transport and the black spots represent prey that are not retrieved yet.

ARGoS makes it possible to use the code of a behavior for the simulated robots on the real robots, that is to say, the simulated robots in ARGoS uses the same commands as the real robots. Because of this, the effort necessary to move from simulation to real robots is low.

ARGoS currently supports the Swarmanoid robots [17] and the e-pucks (Figure 3.7a). It also contains the IRIDIA TAMs (Figure 3.7b). The highly modular architecture of ARGoS allows to easily add new actuators and sensors to the robots and to add multiple implementations of each actuator and sensor.



(a) (b)

Figure 3.7: Images of simulated objects in ARGoS. (a) simulated e-puck in ARGoS (b) simulated IRIDIA TAM in ARGoS.

3.2.2 The e-puck Robots

The robots used in this master thesis are the e-pucks. The *e-pucks* (Figure 3.8a) are mobile robots designed for educational purposes and research [35]. Swarm robotic laboratories require reliable and affordable robots. Researchers also prefer small robots so that the experiments do not require too much space and so that it is possible to develop the robots on their desks. E-pucks have a diameter of 7.5 cm, a height of 6 cm and a weight of 660 g. But despite their small size, they are rich in sensors and actuators.

The different sensors of e-pucks are:

- Eight infrared proximity sensors placed around the body of the e-puck for measuring the proximity of obstacles neighboring the e-puck.
- A 3D accelerometer measuring the acceleration of the e-puck and its inclination.
- Three microphones allowing to localize the source of a sound by triangulation.
- A camera (640x480 pixels) in front of the e-puck.

The various actuators are:

- Two stepper motors, one for each wheel, having a full rotation of 1000 steps per wheel.
- A speaker, which makes communication between e-pucks possible.
- Eight red light emitting diodes placed around the body of the e-puck allowing a visual interface with the user and other e-pucks.
- A set of green LEDs placed in the transparent body.

The memory size (8 kB of RAM and 144 kB of flash memory) and the processing power (64 MHz and 16 MIPS of peak processing power) of the e-puck are limited. Therefore, the e-puck can only grab a part of a picture of the camera and is constrained in the number of sensors and actuators it can process in parallel.

To add more features, it is possible to connect extension parts on e-pucks. In this work, we use one particular extension: the *embedded Linux system board* (Figure 3.8b)[32]. The Linux board adds more processing power, memory and capabilities to the basic e-puck platform. The addition of the Linux board allows the e-puck to fully use the front camera and to process the sensors and actuators in parallel. It also gives a more powerful control architecture: in contrast with the basic e-puck, the extension makes it possible to program in high level languages with the standard Linux tools.

Another extension used in this thesis is the *omnivision turret* (Figure 3.8c). It provides the e-puck with a camera that takes 360 degrees panoramic pictures. The

camera is placed in a plastic cylinder and points upwards to a spherical mirror. It has the same resolution as the one mounted on the basic e-puck, 640x480 pixels.

(a) (b) (c)

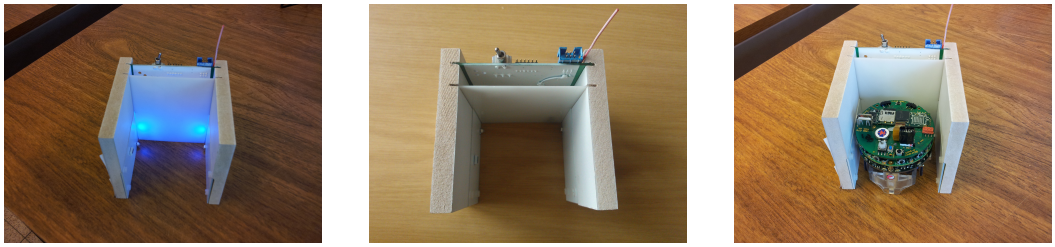
Figure 3.8: Pictures of the e-puck robot and some of his extensions that can be placed on top of the e-puck. (a) a basic e-puck with a speaker extension part (from www.gctronic.com) (b) the Linux board extension part (from www.roadnarrows.com) (c) the omnivision turret extension part (from www.cyberbotics.com).

The experiment of this thesis is performed on simulated e-pucks in ARGoS (Figure 3.7a). All the capabilities of the e-pucks are implemented in ARGoS. For the experiment of this master thesis only the proximity sensors, omnidirectional camera and wheel actuators are used. Since the omnidirectional camera has been developed only recently, it was not available in ARGoS at the beginning of this thesis. One contribution of this thesis was to implement a simulated version of the omnidirectional camera such that it is able to identify the relative angle and distance of colored lamps. The omnidirectional camera sensor of the e-puck is functional in ARGoS but is not yet visible when running the simulator.

3.2.3 IRIDIA TAMs

The e-puck is a limited robot that cannot act on the real world so we need a way to simulate the interaction with the environment. For this reason Brutschy et al. [10] developed the IRIDIA TAM. The point of it is to provide a device able to reproduce a space where a "task" need to be done. A task can be something like picking up an object or even something more complex, like repairing another robot. The tasks represented by the IRIDIA TAM are of course an abstraction, as an e-puck is not capable of tasks that needs physical interaction with the environment. Throught the use of the IRIDA TAM, an e-puck can perform pretended tasks. The LED of the TAM are used to send color signals to the robots, and therefore, to communicate with them by using a color specific to a message. For example, it can be used to display to a robot that there is a task to be done and display that a task is currently under execution.

The TAM is composed of three walls making a U-form when viewed from the



(a) (b) (c)

Figure 3.9: Pictures of IRIDIA TAMs in development at the IRIDIA-ULB lab: (a) the IRIDIA-TAM with the LEDs on (b) top-view of an IRIDIA TAM (c) an e-puck entering an IRIDIA TAM.

top (Figure 3.9). It is a little bit larger and deeper than the diameter of an e-puck. The inside part of the back wall consist of a translucent plastic plate diffusing the light produced by two RGB LEDs situating between the back wall and the plate. The lights of the IRIDIA TAMs can produce a broad array of colors (RGB color model). Additionally, an IRIDIA TAM contains a light barrier so that it is able to detect if an object has entered.

The IRIDIA TAM is equipped with only one sensor:

- A light barrier.

And, it contains also only one actuator:

- A LED actuator, which consist of 2 RGB LEDs.

A drawback of the IRIDIA TAM is that the behavior code of simulated IRIDIA TAMs can not be transfer to the real-world IRIDIA TAMs. However, the controllers of the real-world IRIDIA TAMs are very simple and thus implementing the same behavior can be done easily. The implementation of this feature is currently under development.

3.2.4 PRISM

This thesis is strongly based on model checking. As model checker we use PRISM. *PRISM* is a free and open source probabilistic model checker, that is, a tool for formal modeling and analyzing systems with probabilistic behaviors [27]. PRISM is able to check different types of probabilistic models:

- Discrete-time Markov chains (DTMCs)
- Continuous-time Markov chains (CTMCs)
- Markov decision processes (MDPs)

- Probabilistic timed automata (PTAs)

To use PRISM, the models have to be expressed in the PRISM language, which is a simple, high-level modeling language. PRISM constructs a precise mathematical model of the systems described in the PRISM language. Users can simulate runs of the paths of the probabilistic models (discrete-event simulation engine).

Properties of the system can be analyzed. PRISM supports several types of temporal logic:

- PCTL (probabilistic computation tree logic)
- CSL (continuous stochastic logic)
- LTL (linear time logic)
- PCTL* (which subsumes both PCTL and LTL)

It also incorporates quantitative properties (costs/rewards). It can calculate the percentage of all possible outcomes that satisfy a property, calculate the average value of a quantitative property, verify if a property is always verified after passing a certain point in the model,... In PRISM, model checking can be done exhaustively (also known as complete model checking or simply model checking) or in an approximate way (also known as statistical model checking).

Chapter 4

Formal Verification of Flexibility

In this chapter I present the main contributions of this master thesis. We will see how to apply model checking on a swarm robotic system in order to formally verify flexibility. This chapters builds on the definitions and concepts presented in the previous chapter. Specifically, we will adopt the definition of flexibility given in Section 2.1 and use model checking as presented in Section 3.1.3. I tested it on a specific swarm robotic scenario: foraging. As I will explain later on, foraging has been chosen as a case study due to its generality.

This chapter is organized as follow. Section 4.1 introduces the swarm robotic application on which we verified flexibility. Section 4.2 explains how we implemented the system using the ARGoS simulator. Section 4.3 outlines and compares various possible approaches to model swarm robotic system and criticize them. Finally, Section 4.4 reports and analyzes the different results obtained in simulation and by using model checking.

4.1 Scenario: Foraging

In this work, I have chosen the foraging scenario for multiple reasons. First, it is one of the most common scenarios in swarm robotic literature [24, 31, 33]. Second, the foraging scenario is quite simple, its abstraction of tasks allow us to focus on the global behavior of the system and not on the details of the tasks. Third, it is generic because many scenarios require robots to collect objects from one point and deliver them to another. For example, such a scenario could be a factory creating toys, where the robots would pick up toys from the assembly lines and deposit them in trucks that will carry the toys to the various stores. Finally, the foraging scenario allows to present a scenario in which simple modification to the environment (such as, the arena size, number of tasks and time delays) influence greatly the output of the system, thus making it a good example of the need for flexibility in swarm robotics systems.

I define the foraging system of this work as follow:

> A swarm of robots needs to explore an unknown environment searching for tasks to accomplish. Once a robot completes a task, it returns to the nest to deliver the result. It then proceed to search for another task to complete.

As explained in Section above, in this work, we make use of an abstraction to represent tasks to complete. In particular, the robots do not interact with the environment. Instead, they interact with dedicated modules, called TAMs (Task Abstraction Modules) which are described in Section 3.2.3. The interaction with the TAM is very abstract and consists only of a waiting times. The TAM communicates with the robots through specific colors.

The foraging system described in this section is similar to the foraging system described by Konur and Dixon [24], in which a swarm of robots explore an environment, find and collect prey, and bring them back to a nest area (see Figure 4.1).

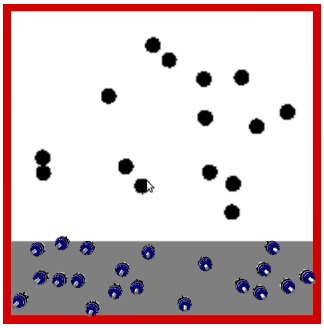
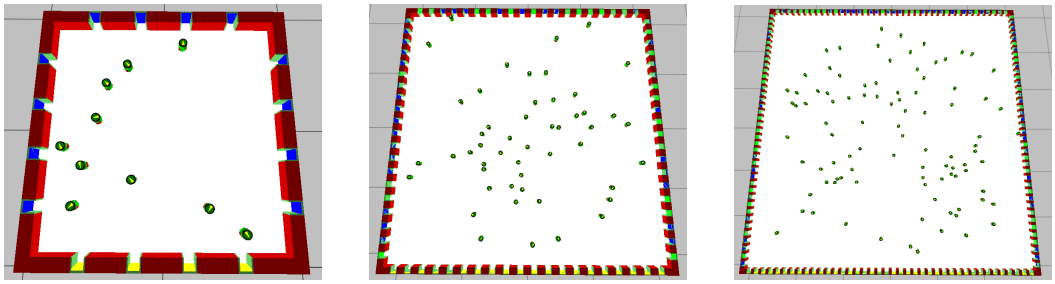


Figure 4.1: The foraging system described by Konur and Dixon [24]. The red walls correspond to the limits of the environment, the black spots represent the prey that the robots, represented by the blue spots, must bring back to the nest, represented by the gray floor.

In the environment there are two kinds of TAMs: in three sides of the environment are available TAMs that represent tasks to tackle, while in the last side there are TAMs that represent the nest. I will refer to the TAMs used for the tasks as the *Task-TAMs* and to the TAMs used for the nest as the *Nest-TAMs*. The environments for the experiments have square forms with Task-TAMs on the top border, on the left border and on the right border (see Figure 4.2). The Nest-TAMs are placed at the bottom of the environment. Note that the Task-TAMs and the Nest-TAMs are composed of the same hardware but run a different software. The TAMs are placed equidistant from each other. At the beginning of the experiment the e-pucks are placed randomly in the environment.



(a) (b) (c)

Figure 4.2: Pictures of the different sizes of the flexibility modeling system in ARGoS (a) the small size environment with 10 e-pucks, 12 Task-TAMs and 4 Nest-TAMs (b) the medium size environment with 50 e-pucks, 60 Task-TAMs and 20 Nest-TAMs (c) the big size environment with 100 e-pucks, 120 Task-TAMs and 40 Nest-TAMs.

4.2 ARGoS simulation

The experiments performed for this thesis have been done using the ARGoS simulator, where the IRIDIA TAMs are already integrated. The behavior of each type of entity, that is, e-pucks, Task-TAMs and Nest-TAMs, are described in separated software, called the controller. The behavior of each entity, in this master thesis, is described using a state machine. The state of an entity changes according to different triggering events: for example, a Task-TAM changes state when a robot enters it.

4.2.1 Behaviors

In this section, I describe in detail the behavior of each entity.

Task-TAMs

The Task-TAMs have four states:

- Free (green): the TAM has a task ready and waits for an e-puck to enter.
- Busy (red): the TAM is currently occupied by an e-puck and waits for the task to be performed.
- Waiting (blue): the TAM waits for the e-puck, that just accomplished the task, to leave.
- Idle (blue): the TAM has no task to be done.

The TAM goes from the free state to the busy state when a robot enters the TAM. To go from the busy state to the waiting state, the TAM sets the busy time randomly according to an exponential distribution where the average time is the same for each

TAM. The TAM stays only for a defined amount of time in the waiting state before changing his state to the idle state. The idle state changes to the free state with a fixed probability each timestep.

Nest-TAMs

The Nest-TAMs have three states:

- Free (yellow): the TAM waits for an e-puck that has accomplished a task to enter.
- Busy (red): the TAM is currently occupied by an e-puck robot and waits for the e-puck to finish delivering the result.
- Waiting (yellow): The TAM waits for the e-puck to leave.

The Nest-TAMs work pretty much the same way as the Task-TAMs except that they do not have an Idle state, that is, they are always ready for a deposit by a robot. They go directly from the waiting state to the free state.

E-Pucks

The e-pucks have five different states. The behavior in those states are a little different when the e-pucks need to go back to the nest to deliver the information about the completed task or when the e-puck is looking for a task (see Figure 4.3). The different states are:

- Random walk (RW): the e-pucks walks randomly in the environment until it finds a free Task-TAM or a free Nest-TAM when it accomplished a task.
- Walk to TAM: the e-pucks walks in the direction of the free TAM until it is at a distance of two times its diameter. If for any reason (for example, another robot already occupied the TAM) it cannot see the TAM anymore, it goes back to the random walk state.
- Enter the TAM: the e-pucks enters the TAM. Again, if the e-puck does not see the TAM anymore, it goes back to the walking randomly state.
- In TAM: the e-pucks waits for the task to be accomplished or waits for the information of the completed task to be delivered and afterwards goes to the leaving state.
- Leave TAM: the e-pucks leaves the TAM by going backwards and thereafter by walking randomly.

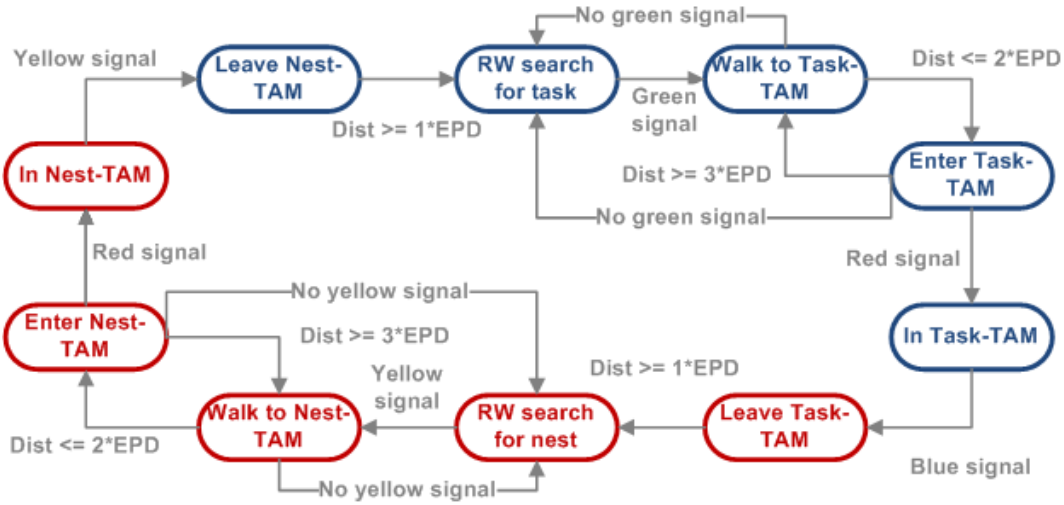


Figure 4.3: Graph of the different states and transitions between the states of the e-puck behavior in the foraging system. A transition with a color signal symbolizes the fact that an e-puck sees a blob (LEDs of the TAM) of that color. *EPD* stands for e-puck diameter and *Dist* stands for the distance between the e-puck and the blob to which he walks. The states where the e-puck has accomplished a task are in red and the other ones are in blue.

The e-puck and TAM communication

The TAMs communicate their state to the e-pucks by changing the color of their LEDs. If an e-puck is close enough to the TAM, it can see a blob, which is a spot of color, produced by the TAM. The e-puck is able to detect the blobs that are less than half a meter away from it and can calculate the distance and direction at which the blobs are.

The Task-TAMs have three different colors: green for the free state, red for the busy state and blue for the idle and waiting state. So when the e-pucks searches for a task, he looks for Task-TAMs with their LEDs on green. If an e-puck has completely entered a Task-TAM, the color of the TAM changes to red. The e-puck waits for the LEDs of the Task-TAM to be blue, which signals that the task is accomplished, and then leaves the TAM.

The Nest-TAMs work in the same way as the Task-TAMs except that there are only two colors: yellow for the free and waiting state and red for the busy state. The color of the free state is different than the one of the Task-TAMs due to the fact that the e-pucks must be able to differentiate the two types of TAMs.

4.2.2 Implementation specifications

The random walk algorithm

The random walk that is implemented for this experiment works as follow. The basic behavior is to go straightforward, but whenever the e-puck encounters an obstacle, that is to say, its proximity sensors have detected an obstacle, it rotates and go straightforward again. According to the measurements of the different proximity sensors, the e-puck decides whether to rotate left or right. More specifically, if the proximity sensors report that there is an obstacle on the left, then the e-puck rotate right and vice versa.

A rotation is done by setting the wheel speeds with opposite values and letting the e-puck rotate for a random number of timesteps. A rotation of one timestep in the experiment is equal to approximately a rotation of 20 degrees. The number of timesteps the e-pucks rotate is chosen randomly in a range of values going from 1 to 8 (20° to 160°). In this experiment, I used a square function in order to increase the probability of a small rotation. This is to avoid turning to far and having to return in the opposite direction and to limit the number of times that the e-puck goes back were it comes from. The formula that I used is:

$$Number\ of\ timesteps = Max + Min - \lceil \sqrt{rand(Min \times Min, Max \times Max)} \rceil$$

How the e-puck enters a TAM

Whenever a e-puck detects a free TAM during a random walk it perform the following operations:

- It turns until the angle between its direction and the position of the blob (light produced by the LEDs of a TAM) is smaller than a certain threshold. Afterwards, the e-puck goes straightforward until it is at twice its diameter from the TAM's LEDs. Whenever the angle exceeds the threshold, the e-puck changes its direction in order to realign with the TAM. In the case that the e-puck encounters an obstacle, it goes in the random walk state for some timesteps.
- At a distance smaller than twice the e-puck diameter, the e-puck enters the TAM with a velocity depending on the values it receive from the proximity sensors. This is necessary because the arrival angle relative to the TAM's entry is not always the same.
- The e-puck stops when the TAMs color changes to red (busy state) and the LED's of the TAMs are less than $\frac{1}{5}$ of its diameter away from the e-puck.

The visibility range of the omnivision turret camera of the e-pucks is of $0.5m$, therefore the e-puck only sees the blobs that are at maximum $0.5m$. When there

are multiple blobs of the same color, the e-puck only takes into account the closest one.

4.3 Model Checking

In section 3.1.3, we saw that a practical way to formally verify properties of a swarm robotic system is model checking. Section 3.1.3 also concluded that CTMCs are better in terms of performance than DTMCs. In this section, two different CTMC models of the foraging scenario are presented. The first one tries to resemble as much as possible to the simulated system in ARGoS, whereas the second one is a more abstract model of the system.

In order to make the creation of the CTMCs easier, I start by transforming all the transition conditions of the e-pucks behavior graph from Figure 4.3 into constant time or stochastic time conditions (see Figure 4.4). Constant time (represented by T) is here considered as an amount of time which may vary a little but that is still bounded. For example, when an e-puck is walking to a TAM that it has detected, the time for it to arrive is normally fixed. A distinction is made between two types of stochastic time conditions, those that are limited by constant times (represented by P) and those that are not (represented by R). In the first case, they can be seen as the probability of not taking the transition with a constant time condition. For instance, during the time an e-puck walks to a TAM, the TAM can be taken by another e-puck and therefore must restart searching for a new free TAM. In the second case, the time can theoretically go from 0 up to an infinite number of timesteps, such as when an e-puck is searching for a free TAM, there is no guarantee about the time it will take.

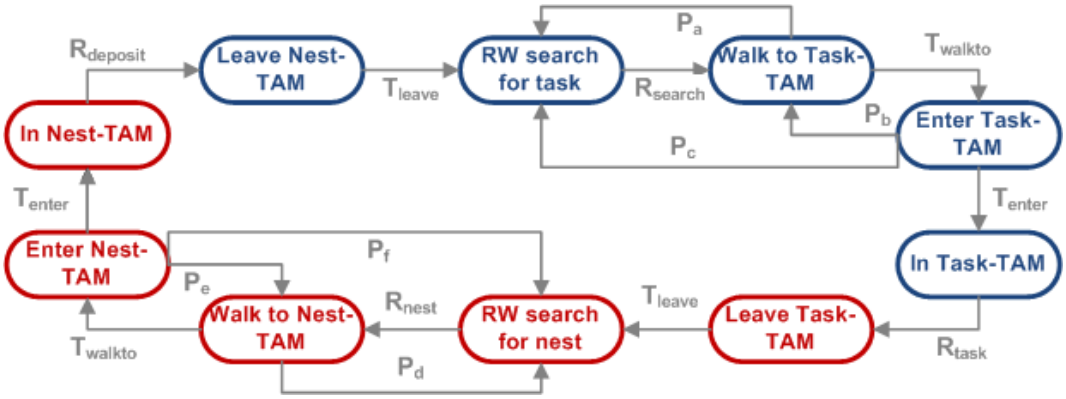


Figure 4.4: Graph of the different states and transitions between the states of the e-puck behavior in the foraging system. The transition's conditions of the graph are expressed in constant and stochastic times. The transitions where the duration of an action is fixed, are represented with the T symbol, the transitions where there is a probability that the e-puck takes another transition in a fixed amount of time are represented with the P symbol and the transitions where the duration of an action takes a stochastic time are represented by the R symbol.

4.3.1 First model

The first CTMC model we present is the closest to the simulated version of our system. For this reason, it is also the most complex one. Since it is not possible to reproduce all the details of the simulated system, some simplification have been done. I merged the two states *walk to Task-TAM* and *enter Task-TAM*. These two states, as it is possible to see in Figure 4.4, are logically related as their merge will not modify the rest of the graph. The duration of the time needed by an e-puck for being in a booth after having seen that it is free, is equal to the addition of the time for an e-puck to walk to the booth and the time taken to enter the booth. The same reasoning can be applied on the *walk to Nest-TAM* and the *enter Nest-TAM* states.

Since CTMC only accepts rates as transition conditions between states, as seen in section 3.1.2, each transition condition of the graph of Figure 4.4 must be transformed in rates. That can be done thanks to the formulas described in section 3.1.3. Due to the fact that fixed times are converted to rates, we can remove the probability conditions. Indeed, rates cannot represent fixed times, however thanks to the fact that there are probabilities going to previous states, the time to go to a next state can vary and can then be considered as stochastic. For instance, when a e-puck walks to a TAM, it is possible that at a moment it does not see the blob anymore and restarts searching for a TAM. The time the e-puck takes to walk to a TAM is then not fixed anymore.

To go from the graph of Figure 4.4 to the graph of the first model shown in Figure 4.5a, the following steps must be done:

- Remove all the P transitions.
- Merge the *walk to Task-TAM* state with the *enter Task-TAM* state and the *walk to Nest-TAM* state with the *enter Nest-TAM* state.
- Merge the *in Task-TAM* state with the *leave Task-TAM* state and the *in Nest-TAM* state with the *leave Nest-TAM* state.
- Compute all the transition rates.

The first two steps are explained in the previous paragraphs. The third step can be done by adding the fixed time for an e-puck to leave a TAM to the average time for an e-puck to find a task and to the average time to find the nest. This mean that we consider the leaving time of the e-puck as searching time.

The different rates of the first model CTMC are calculated as follow:

- $R_{searching} = \frac{-ln(1-P_{find\_task})}{1}$, where P_{find_task} is the probability of an e-puck finding a free Task-TAM after one second. P_{find_task} is calculated as follow:

$$P_{find\_task} = \alpha \times \frac{S_{TAMs}}{S_{area}}$$

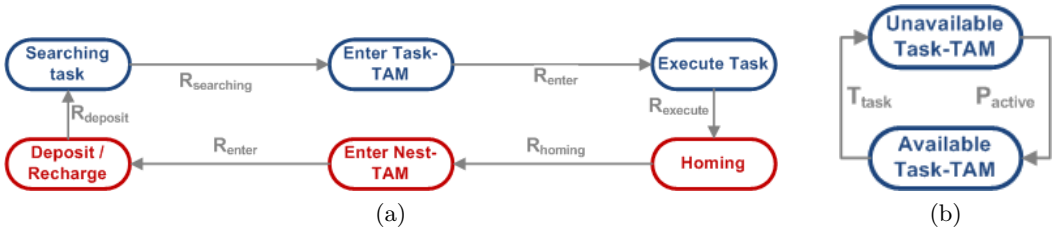


Figure 4.5: CTMCs of the first model (a) the behavior of the e-pucks (b) the behavior of the Task-TAMs.

where S_{TAMs} is the size of the surface from which an e-puck can see a Task-TAM, S_{area} is the environment surface and α is a proportional factor.

- $R_{enter} = 1 - e^{-1/t_{enter}}$, where t_{enter} is the average time in seconds for a robot to enter a free TAM after he sees one.
- $R_{execute} = 1 - e^{-1/t_{execute}}$, where $t_{execute}$ is the average time to execute a task.
- $R_{homing} = \frac{-ln(1-P_{find\_nest})}{1}$, where P_{find_nest} works in the same way as P_{find_task} but the Task-TAMs are replaced in the formula by the Nest-TAMs.
- $R_{deposit} = 1 - e^{-1/t_{deposit}}$, where $t_{deposit}$ is the average time that an e-puck stays in a Nest-TAM.

These are the rates for the CTMC of a single robot. As seen in Section 3.1.3, to construct a CTMC of the global system with the macroscopic approach, a counter must be added in each state. This counter retains the number of e-pucks in the corresponding state. Additionally, each transition rate must be multiplied by the number of e-pucks that are in the outgoing state of the transition.

It must be noted that, not only some rates of the models depends on the number of e-pucks currently in a specific state, but some rates depend also on the number of free/occupied TAMs. Computing the number of free Nest-TAMs can be done by subtracting the number of e-pucks in the depositing state from the number of Nest-TAMs. However, computing the number of free Task-TAMs is more complicated than the Nest-TAMs due to the fact that they can be unavailable because no task is ready. Therefore, a CTMC, with two states, for the Task-TAMs is added. One state is used to count the number of Task-TAMs that are currently unavailable and the other counts the ones that are available (see Figure 4.5b). A TAM becomes available with some apparition rate and becomes unavailable if a e-puck enters. The availability rate of one Task-TAM is multiplied by the number of Task-TAMs in the unavailable state.

The value of the different parameters are discussed in section 4.4.

The problem of the first model is that its state space grows too fast. Table 4.1 shows us that the state space is already too big for probabilistic model checking with PRISM, which is able to perform complete model checking of system composed of

maximum 10^7 states, when the environment size is medium (see Section 4.4 for more explanations about the environment sizes). The formula to compute the number of states with the first model is:

$$\left(\binom{N_{epucks}+K-1}{N_{epucks}} - \binom{N_{epucks}+K-1-N_{nests}}{N_{epucks}-N_{nests}}\right) \times \binom{N_{tasks}+K_2-1}{N_{tasks}}$$

where N_{epucks} is the number of e-pucks in the experiment, K is the number of states in which an e-puck can be in (in this model K equals 6), N_{nests} is the number of Nest-TAMs, N_{tasks} is the number of Task-TAMs and K_2 is the number of states in which the Task-TAMs can be in (in this model K_2 equals 2). The second combinatorial term of the formula is introduced because the number of robots in the *in Nest-TAM* state is bounded due to the number of Nest-TAMs being smaller than the number of e-pucks.

	Small	**Medium**	**Big**
First model	$\sim 3.3 \times 10^4$	$\sim 2.1 \times 10^8$	$\sim 10^{10}$
Second model	202	$\sim 1.8 \times 10^4$	$\sim 1.3 \times 10^5$

Table 4.1: Comparison of the number of states of the two different CTMC approaches in the three different environment sizes.

4.3.2 Second model

To solve the state explosion problem, I constructed a second model with a reduced number of states. The first simplification is to remove the Task-TAM CTMC, which is an important factor in the space state explosion of the first model. The average number of available Task-TAMs can be estimated from simulation executions of ARGoS. The second simplification is to merge the *searching* and the *enter Task-TAM* states and to merge the *homing* and the *enter Nest-TAM* states. The time needed by an e-puck to enter a TAM is normally constant if there are no exceptions, such as another e-puck entering the TAM before it. Therefore, adding the entering time to the average searching can be done and make sense.

Figure 4.6 shows the obtained CTMC after the simplifications. The transition going from the *searching* state to the *execute task* state symbolizes then the time an e-puck would take to find a Task-TAM and to enter it. The same can be said about the transition going from the *homing* state to the *deposit* state.

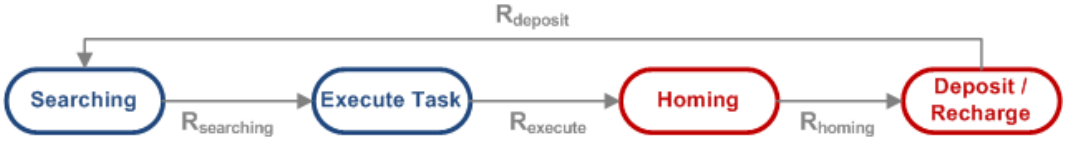


Figure 4.6: CTMC of the second model

The different rates of the second model are calculated as follow:

- $R_{execute}$ and $R_{deposit}$ are computed in the same way as in the first model.
- $R_{searching}$ and R_{homing} are values that depends on many different aspects of the system (such as the number of robots in each state, the number of available TAMs, the size of the environment, ...). Since computing this values analytically would be very complex, I decided to estimate them from simulation executions of ARGoS.

By looking at Table 4.1, we can see that the number of states is much smaller with the second model, than with the first model presented in the previous section. It is possible to even verify flexibility in the big environment with probabilistic model checking. The formula to compute the number of states with the second model is:

$$\binom{N_{epucks} + K - 1}{N_{epucks}} - \binom{N_{epucks} + K - 1 - N_{nests}}{N_{epucks} - N_{nests}}$$

where the symbols have the same meaning than those of the states space computation of the first model (but in this case K equals 4).

4.4 Results

In this work, the different parameters that are used to verify the flexibility of the foraging system are:

Size: This parameter deals with the area of the environment, the number of robots and the number of Task- and Nest-TAMs. Those variables are closely correlated, that is to say they increase and decrease proportionally one to the other (see Figure 4.2). The formulae used to calculate the values of each variable of a specific size are defined as follow: area size equals $4 * p$, number of robots equals $10 * p$, number of Task-TAMs equals $12 * p$ and the number of Nest-TAMs equals $4 * p$, where $p \in \Re_{\geq 0}$ is a proportional factor. I analyzed the system using three different configuration sizes. The first configuration (small size) has a size length of $2m$ (area of $4m^2$), 10 robots, 12 Task-TAMs and 4 Nest-TAMs. The second configuration (medium size) has a size length of $4.5m$ (area of $20m^2$), 50 robots, 60 Task-TAMs and 20 Nest-TAMs. The third configuration has a size length of $6.3m$ (area of $40m^2$), 100 robots, 120 Task-TAMs and 40 Nest-TAMs.

Task duration: This parameter specifies the time necessary for a robot to complete a task, that is, the time a robot must wait in the TAM. I analyzed the system using the following values: 5s and 50s.

Time in nest: This parameter specifies the time necessary for a robot to report the result of the task in the nest, that is, the time that a robot needs to spend in the Nest-TAM. The values analyzed for this parameter are 5s and 50s.

Appearance rate: This parameter specifies the rate at which a Task-TAM becomes available. The values tested for the rate of this parameter are 0.01, 0.001 and 0.0001.

The total number of experiments with distinct parameters is equal to $\#Size \times \#Task\_duration \times \#Time\_in\_nest \times \#Appearance\_rate = 36$.

This section starts by first analyzing the results obtained using the ARGoS simulator. Afterwards, I analyze the results of the two PRISM models and verify if they are representative of the simulations done in ARGoS.

4.4.1 ARGoS simulations

To have a significant sample of each experiment, each distinct experiment was run 100 times. This gives us a total of 3600 experiments to run. The experiments were executed on Majorana, the computer cluster of Iridia, that contains 15 computional nodes 2-Core AMD Opteron 2216 HE working at 2.4GHz and 4GB of RAM, 32 computational nodes 4-Core INTEL Xeon E5410 working at 2.33GHz and 8GB of RAM, 16 computational nodes 8-Core AMD Opteron 6128 working at 2GHz and 16GB of RAM and 16 computational nodes 16-Core AMD Opteron 6272 working at 2.1GHz and 64GB of RAM. The execution time of all the experiments took about one hour.

The number of parameters of the system is too high to plot them all on one figure, since a five dimension graphic would be needed (4 parameters plus the number of accomplished tasks). Therefore, I chose to show different "slices" of the results, presenting the most interesting parameters one at the time. In particular, if we want to analyze the flexibility of the foraging system, the number of accomplished tasks must be present in each graphic. It is the resulting parameter of the foraging scenario that reports the performance of the system.

Figure 4.7 shows the box-plot of the number of accomplished tasks in function of the different sizes and for the three appearance rate values. In Figure 4.7, a figure is drawn for each possible combination of values of the *task duration* and the *time in nest* parameters. This allows us to have a visualization of all the obtained results of the different experiments. The four graphs report that the number of accomplished tasks grows almost linearly according to the size of the experiment. One could have expected that the number of accomplished tasks was correlated to the number of robots, however this is not the case due to the fact that average distance that the e-pucks must travel grows with the size of the environment.

Clearly, the appearance rate has a significant effect on the performance. The smaller the rate, the smaller the number of accomplished tasks becomes. When the appearance rate is equal to 0.0001, the performance of the system does not change significantly with respect to the time needed for a task or for deposit the result to the nest. This is due to the fact that the average time for a Task-TAM to become available is so big that the duration of the other actions becomes irrelevant. When the

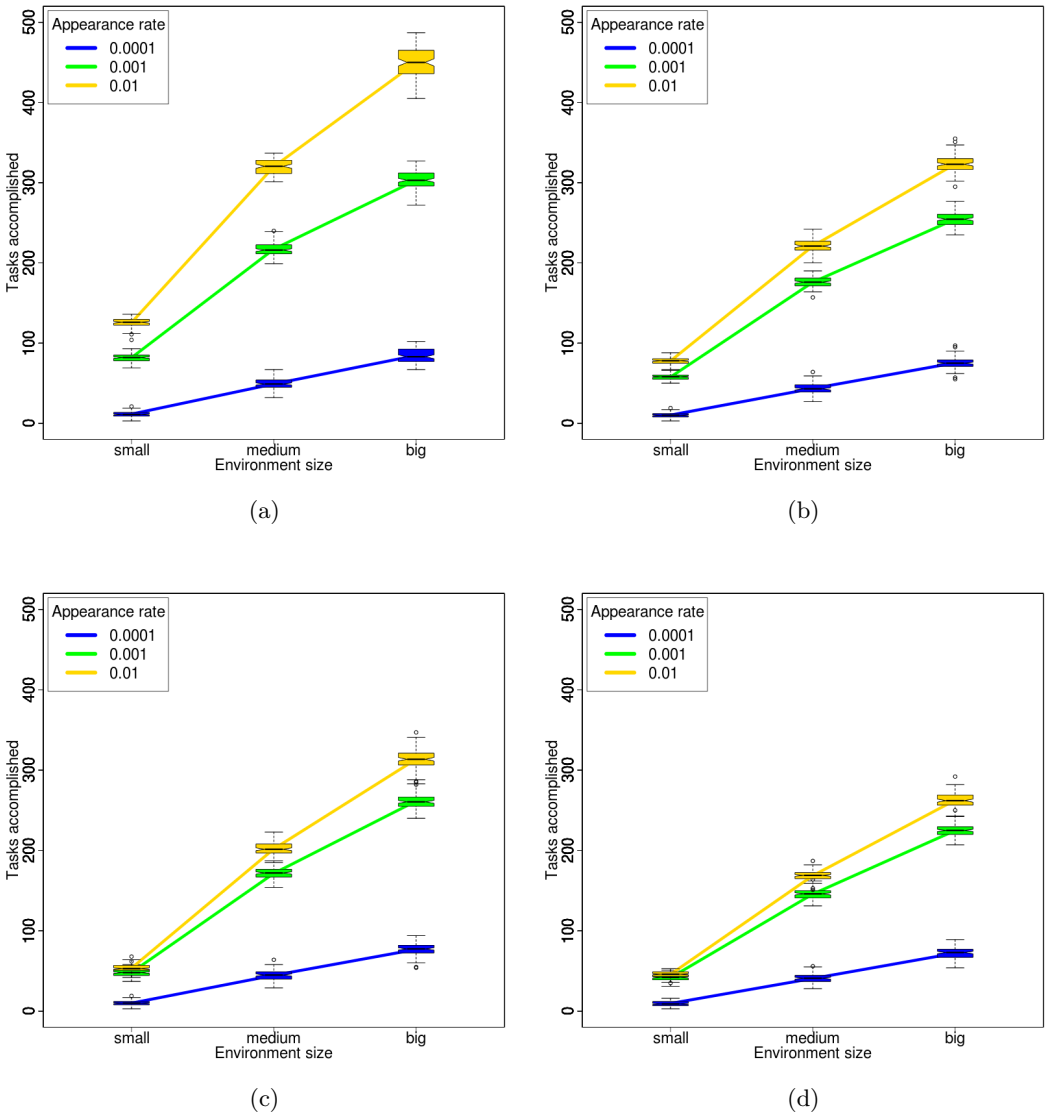
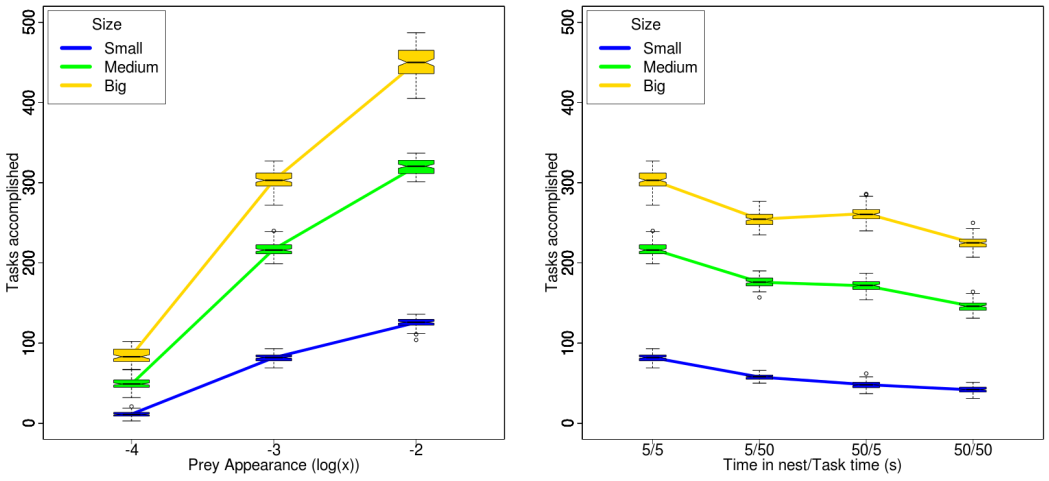


Figure 4.7: box-plot graphs of the number of accomplished tasks in function of the different environment sizes and that for the three appearance rate values. (a) the parameters *task duration* and *time in nest* are set at 5s (b) the parameter *task duration* is set at 50s and the parameter *time in nest* is set at 5s (c) the parameter *task duration* is set at 5s and the parameter *time in nest* is set at 50s (d) the parameters *task duration* and *time in nest* are set at 50s.

appearance rate is equal to 0.001, the different TAM times do not affect significantly the performance. However, the TAM times noticeably affects the performance of the systems where the appearance rate is equal to 0.01. Therefore, the performance of the system with an appearance rate equal to 0.01 comes closer to the performance of the system with a appearance rate equal to $0,001$ when the average times of staying in the TAMs becomes longer.

Figure 4.8a and Figure 4.8b are added here for analyzing the system from other



(a) (b)

Figure 4.8: box-plot graphs of the number of accomplished tasks in function of (a) the different appearance rates and that for all the environment sizes (the task duration and the time in nest were set at 5s) (b) the different combinations of the possible task durations and time in nest values and that for the three environment sizes. Each experiment was run 100 times in ARGoS.

points of view, that is to say by highlighting other variables. The first figure is a box-plot graph of the number of accomplished tasks in function of the appearance rates for the three environment sizes. The average task time and the average result deposit time were set at 5s. The second figure is also a box-plot graph of the number of accomplished tasks in function of the combinations of the values of the the *task duration* and the *time in nest* parameters. The appearance rate was set at 0.001.

We can see that the graphs of Figure 4.8a are almost linear and can consequently conclude that the graphs follow logarithmic trends in function of the appearance rate. Figure 4.8b depicts how the average time for a task or in nest affects the performance of the system. As foreseeable, when the duration of the tasks becomes bigger, the system accomplishes less tasks. We can also see that the average time of the tasks or in nest affect the performance almost in the same way.

4.4.2 Model Checking

In order to verify if the models corresponds to the ARGoS simulation, I statistically model checked each experiment. It was done with PRISM on a HP Pavilion dv6 (Ubuntu 12.10) working at 2.40GHz(4CPUs) and 4GB of RAM. The statistical model checking was run with the default parameters of PRISM, that is to say, with the confidence interval method with a confidence of 0.01, a number of samples equal to 1000 and a maximum path length of 10000. The continuous stochastic logic formula used for verifying the flexibility of the two models is $R =?[C <= 1000]$.

To assure the comprehension of the graphs of this section, I gave each experiment a number. Figure 4.9 shows how the experiments numbers are calculated according to their parameters and Table 4.2 gives an overview of the parameters values of each experiment number.

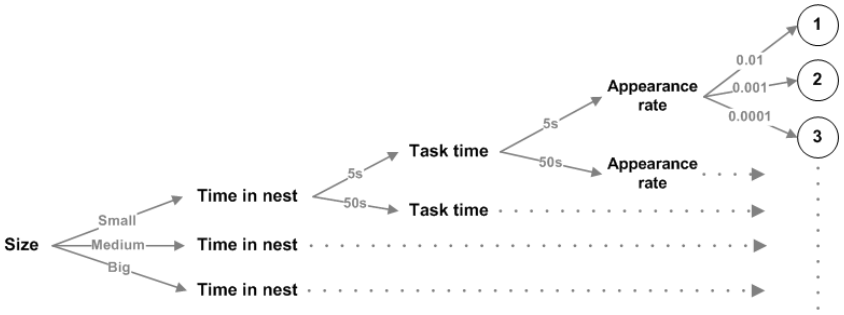


Figure 4.9: Graph showing the corresponding number of an experiment depending on its variable values.

# Exp	Env. Size	Time in nest	Task duration	Appearance rate
1	small	5s	5s	0.01
2				0.001
3				0.0001
4			50s	0.01
5				0.001
6				0.0001
7		50s	5s	0.01
8				0.001
9				0.0001
10			50s	0.01
11				0.001
12				0.0001
13	medium	5s	5s	0.01
14				0.001
15				0.0001
16			50s	0.01
17				0.001
18				0.0001
19		50s	5s	0.01
20				0.001
21				0.0001
22			50s	0.01
23				0.001
24				0.0001
25	big	5s	5s	0.01
26				0.001
27				0.0001
28			50s	0.01
29				0.001
30				0.0001
31		50s	5s	0.01
32				0.001
33				0.0001
34			50s	0.01
35				0.001
36				0.0001

Table 4.2: Parameters value for each experiment number.

First Model

Before running the model, the values of the parameters of the different experiments must be computed using the formulae from Section 4.3.1. They are computed as follow:

- $R_{enter} = 1 - e^{-1/t_{enter}}$, where t_{enter} is set as 5.0 seconds which is the average time necessary for an e-puck to enter a booth. R_{enter} is thus equal to 0.181.
- $R_{execute} = 1 - e^{-1/t_{execute}}$ and $R_{deposit} = 1 - e^{-1/t_{deposit}}$ are computed by replacing $t_{execute}$ by the task duration of the experiment and $t_{deposit}$ by the time in nest duration of the experiment. It should be noted that time is expressed is seconds whereas the parameters in the ARGoS simulator are expressed in timesteps. In order to have the corresponding value of the parameters values in seconds, the value must be divided by 10.
- $R_{booth\_available}$, the rate of a booth availability is equal to the appearance rate multiplied by 10. This is due to the fact that the appearance rate is also expressed in timesteps in the ARGoS simulator.
- $R_{searching} = -ln(1–P_{find\_task})$, where

$$P_{find\_task} = \alpha \times \frac{S_{Task-TAMs}}{S_{area}}$$

where $\frac{S_{Task-TAMs}}{S_{area}}$ represent the proportion of the area in which an e-puck can see an available Task-TAM and α is a proportional factor. More information about α is given bellow. The formula that I use for computing $S_{Task-TAMs}$ is

$$S_{one\_TAM} + \frac{N_{Task-TAMs}}{N_{total\_Task-TAMs}} \times (S_{all\_Task-TAMs} - S_{one\_TAM})$$

where $N_{Task-TAMs}$ is the number of available Task-TAMs, $N_{total\_Task-TAMs}$ is the number of Task-TAMs in the experiment, S_{one_TAM} is the area in which e-pucks can see a TAM (which I will call visibility area of a TAM from now on) and $S_{all\_Task-TAMs}$ is the area in which e-pucks can see at least one Task-TAM when all the Task-TAMs are available. The Task-TAMs visibility areas overlap when they are next to each other, this reduces the maximum area in which a robot may see an available Task-TAM. Using a function allows us to take this reduction into account. In order to limit the complexity of the model, I decided to approximate the visibility area of the available Task-TAMs ($S_{Task-TAMs}$) with a linear function depending on $N_{Task-TAMs}$. S_{one_TAM} (see Figure 4.10a) is obtained with the formula

$$\frac{r^2\pi}{\left(\frac{360}{A_{TAM}}\right)}$$

where r is the visibility range of an e-puck and A_{TAM} is the TAM aperture

angle. A_{TAM} is computed as follow

$$2\ arctan(\frac{W/2}{D})$$

where D is the depth of a TAM and W is the aperture length of a TAM. $S_{all\_Task-TAMs}$ (see gray area of Figure 4.10b) is obtained with the formula

$$1.5 \times L_{env} - 0.5$$

where L_{env} is the length of one size of the environment.

- $R_{homing} = -ln(1–P_{find\_nest})$, where

$$P_{find\_nest} = \alpha \times \frac{S_{Nest-TAMs}}{S_{area}}$$

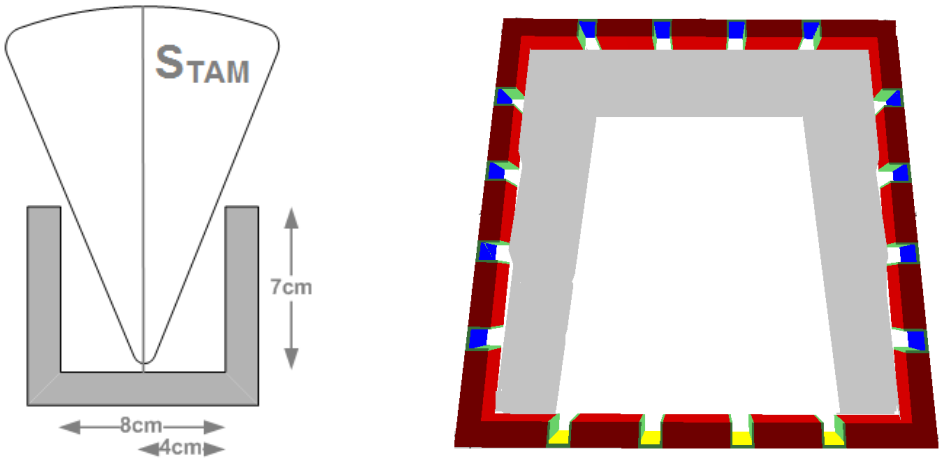
where $\frac{S_{Nest-TAMs}}{S_{area}}$ represent the proportion of the area in which an e-puck can see an available Nest-TAM. The formula that I use for computing $S_{Nest-TAMs}$ is

$$S_{one\_TAM} + \frac{N_{TAMs}}{N_{total\_Nest-TAMs}} \times (S_{all\_Nest-TAMs} - S_{one\_TAM})$$

where $N_{Nest-TAMs}$ is the number of available Nest-TAMs, S_{one_TAM} is the visibility area of one TAM, $N_{total\_Nest-TAMs}$ is the number of Nest-TAMs in the experiment and $S_{all\_Nest-TAMs}$ is the visibility area of all the Nest-TAMs when they are all available. $S_{all\_Nest-TAMs}$ is computed with the formula

$$0.5 \times L_{env}$$

where L_{env} represent the length of one side of the environment.



(a) (b)

Figure 4.10: (a) Visibility area (S_{TAM}) of a TAM (b) maximum area covered by Task-TAMs in the small environment (area in gray).

I computed the value of α using the results obtained using ARGoS. The value was computed for various experiments, but for example purposes I only show it here on the 13th experiment, which is a medium size experiment where the task's appearance rate is set at 0.01 and the task duration and time in nest are set at 5s. The values that are needed to compute α are:

- The average number of Task-TAMs that are available and the average number of Nest-TAMs that are available. Those values were found by running one time the 13th experiment. The average of the number of Task-TAMs that were available at each timestep was equal to 53 and the average number of Nest-TAMs was equal to 18.
- The average time that an e-puck needs to find a free Task-TAM and the average time that an e-puck needs to find a free Nest-TAM. The values I obtained were 476 timeteps for finding a task and 1056 timeteps for finding a nest. More information about the way to find those values is given in the next section.

At first, we have to compute the corresponding probability (over one second) of the average searching times. This can be done by merging together the rate formulae from Section 3.1.3:

$$p = 1 - e^{-1+e^{-1/t}}$$

where t is the average searching time in seconds for a Task- or Nest-TAM. The probability for finding a task in one second is then 0.0259 and the probability for finding a nest in one second is then 0.00937.

Afterwards, we can compute

$$P_{find\_task} = 0.0259 = \alpha \times \frac{(0.13 + \frac{N_{TAMs}}{N_{total\_TAMs}} \times (1,5 \times L_{area} - 0.5 - 0.13))}{S_{area}}$$

$$\Leftrightarrow 0.0259 = \alpha \times \frac{(0.13 + \frac{53}{60} \times (1,5 \times 4.5 - 0.5 - 0.13))}{20.25}$$

$$\Leftrightarrow \alpha = 0.09$$

and

$$P_{find\_nest} = 0.00937 = \alpha \times \frac{(0.13 + \frac{N_{TAMs}}{N_{total\_TAMs}} \times (0.5 \times L_{area} - 0.13))}{S_{area}}$$

$$\Leftrightarrow 0.00937 = \alpha \times \frac{(0.13 + \frac{18}{20} \times (0.5 \times 4.5 - 0.13))}{20.25}$$

$$\Leftrightarrow \alpha = 0.09$$

The α value for P_{find_task} and for P_{find_nest} are the same, which makes it possible to use the same value in the model. Additionally, the value of α is also the same for

all the other experiments, this allows us to set α at 0.09 for all the experiments and therefore we do not have to compute α each time we want to model an experiment with new parameter values.

I should note that α could be found by another way that only depend on the expected number of accomplished tasks of the ARGoS simulations and not on the average number of available Task- and Nest-TAMs and the average time to find a Task and a Nest. We could have tried a range of values for α until the obtained result was equal to the one of the ARGoS simulations. I preferred the first approach because it gives a meaning to the value of α. However, both methods are acceptable.

We now have all the values characterizing the model and it is thus possible to perform model checking on it. The statistical model checking took 408s and the width of the confidence interval was 0.586 (the PRISM file and the shell script to run the experiments can be seen in Appendix A.2.1). It is now necessary to validate the results of the model using the experimental results. I compared the results of all the experiments obtained using statistical model checking with those obtained using the ARGoS simulator. This can be seen in Figure 4.11, where the expected number of accomplished tasks obtained using PRISM are drawn against the box-plot of the simulated results. The width of the confidence interval of the results of the model was expressly omitted due to the fact that it was too small to be seen on the graph.

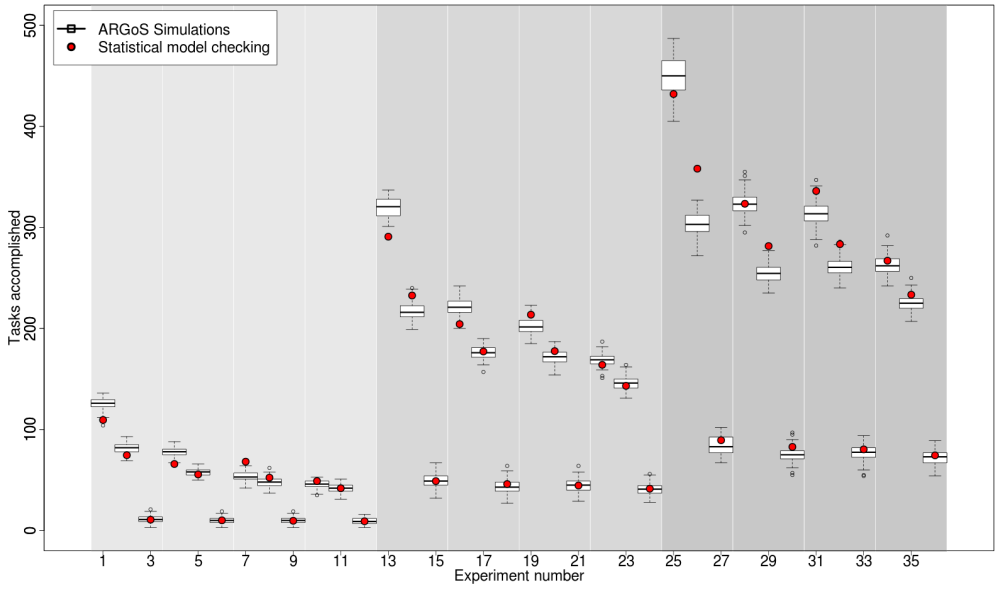


Figure 4.11: Comparison between the results of each experiment (in number of accomplished tasks) obtained using ARGoS and through the first model with statistical model checking. The light gray background represents the experiments done in the small environment, the medium gray represents those done in the medium environment and the dark gray represents those done in the big environment. The white lines group the experiments with the same task duration and time in nest.

By analyzing Figure 4.11, we can see that most of the results of the model are close to those obtained using the ARGoS simulator. The model seems to be a good representation of the ARGoS simulations. Nonetheless, we have to keep in mind that the results obtained using ARGoS are retrieved from a sample of experiments and can therefore also be distant from the real average.

Second Model

For the second model, the exact average time for an e-puck to find a task and the exact average time for it to go back to the nest after accomplishing a task must be found for each different experiment we want to model. Therefore, multiple simulations for each experiment were run in the ARGoS simulator in order to find those values. The problem is that running an experiment for only 10000 timeteps would give erroneous result for the experiments with an apparition rate of 0.0001. This is due to the fact that some of the e-pucks do not find a task to perform in less than 10000 timeteps and by consequence these robots are not taken into account in the statistics. The obtained average time is then much smaller than the real value. To tackle this problem, I ran the experiments for 50000 timesteps, which is enough time for all the e-pucks to accomplish at least one task and gives me more data to fit the parameters of the model.

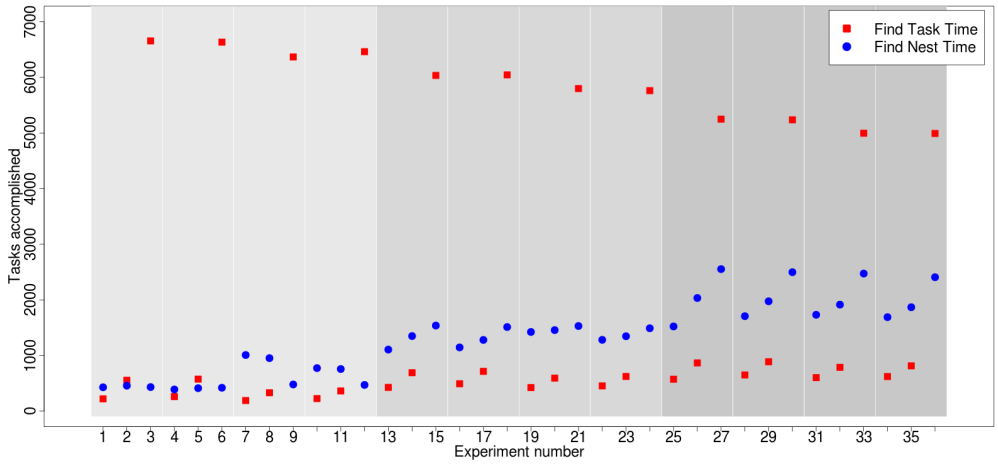


Figure 4.12: Graph showing for each experiment (x-axis) the mean times in timesteps (y-axis) taken by an e-puck to find a task (red squares) and to find the nest after accomplishing a task (blue dots). The background colors have the same meaning as in Figure 4.11.

The different average times of finding a task or a nest for each experiment obtained using the ARGoS simulator are showed in Figure 4.12. The graph reports that the time to find a task is, as expected, really influenced by the appearance rate. The graph also shows that the time needed for going back to the nest is mostly influenced by the size of the arena and the number of e-pucks in the Nest-TAMs. However, in the big size environment, the time needed to find a nest seems influ-

enced by the appearance rate. It is very likely that due to the fact that the e-pucks do not find tasks quickly, the interference between them is much higher.

The point of this model was to be able to use probabilistic model checking to formally verify flexibility. The problem is that even tough the model is small enough to be verified with probabilistic model checking, the last nine experiments take too much time to be checked. Indeed, after then hours of execution, none of the results were found. The result of the 27 experiments that did not take too much time are showed in Figure4.13. The probabilistic model checking of all 27 experiments took 1317s. Due to the fact that this method is very time demanding, I also used statistical model checking on the second model.

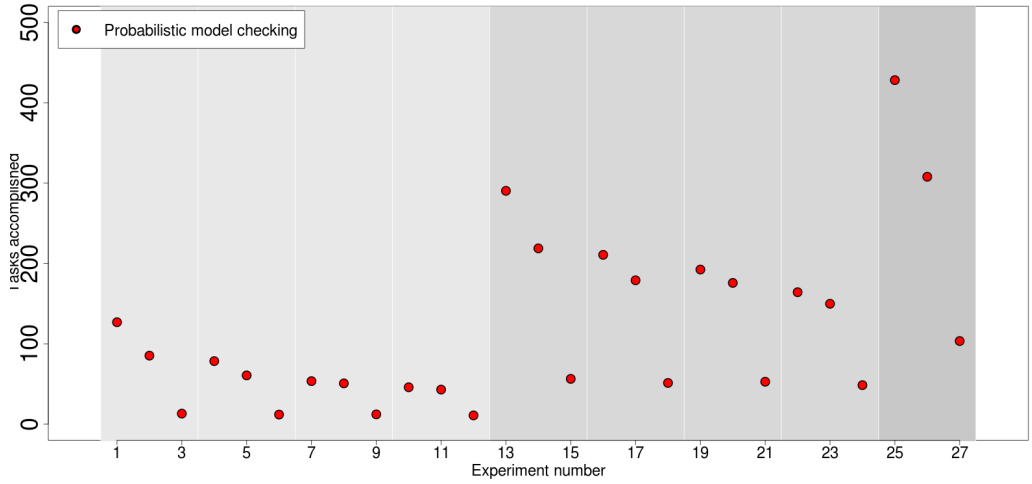


Figure 4.13: Expected number of accomplished tasks of the 27 first experiments obtained using probabilistic model checking on the second model. The background colors have the same meaning as in Figure 4.11.

The statistical model checking on the second model was done in 128s and the average width of the confidence interval was 0,43 (the PRISM file and the shell script to run the experiments can be seen in Appendix A.2.2). The average width confirm us that the results obtained using statistical model checking are really close to those obtained using probabilistic model checking. As with the first model, we want to be sure that the model is representative of the system. Figure 4.14 depicts the number of accomplished tasks of all the ARGoS simulated experiments in box-plot format and the average number of accomplished tasks obtained through the second model. Same as with the first model, if the results for each experiment of the model is in the box or between the whiskers of the box-plots, then we can consider that the results are satisfactory. In this case, only a few of the results are on the limit of the whiskers and the rest are close to the average of the simulations.

The results, except the ones of the experiments with an appearance rate of 0.0001, are better with the second model than with the first model. The reason why the results are not so good with the second model for the experiments with an appearance rate of 0.0001 is due to the fact that the ARGoS simulations were

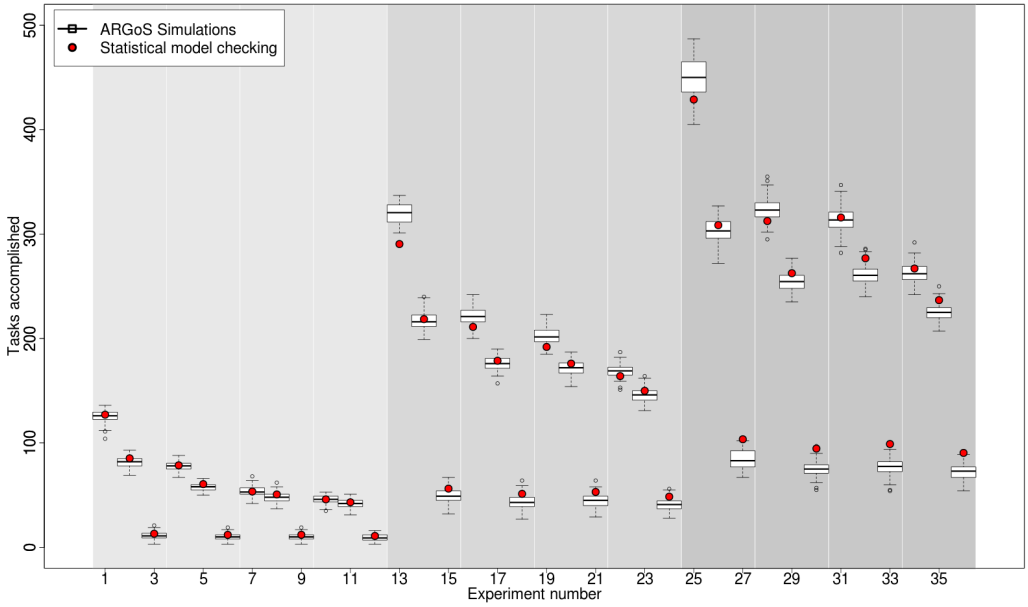


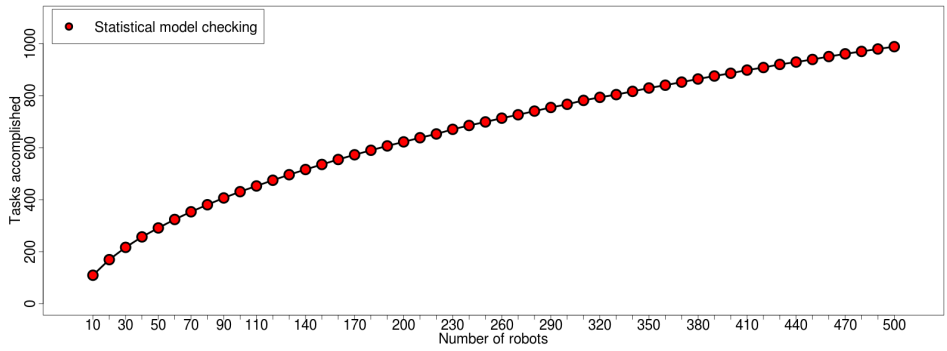
Figure 4.14: Comparison between the results of each experiment (in number of accomplished tasks) obtained using ARGoS and through the second model with statistical model checking. The background colors have the same meaning as in Figure 4.11.

stopped after 50000 timesteps. Therefore some e-pucks that did not found a task in those 50000 timesteps were not taken into account when computing the average time needed to finding a task. This is the main problem of the second model, it has to estimate the average time of finding a task and a nest for each different experiment. Additionally, the same must be done for each new experiment. The first model only need a few simulated experiments in the ARGoS simulator to compute the value of α. This make the strength of the first model. However, the second model is much faster when using statistical model checking and can even use probabilistic model checking with a lot of experiments.

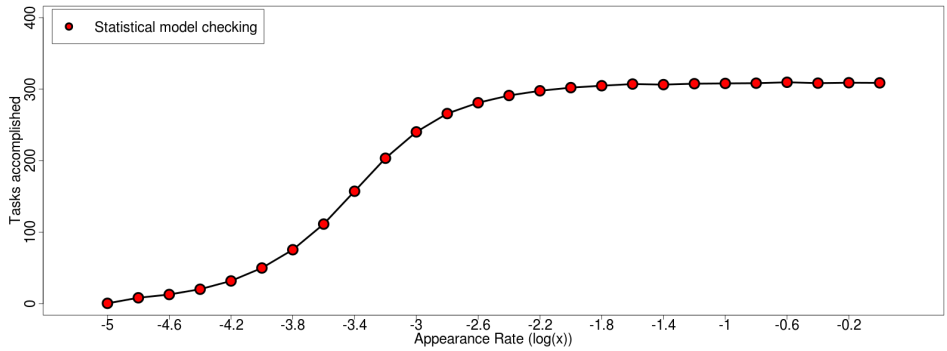
4.4.3 Prediction

In this section, I use the model to perform analysis of the system outside the parameter space analyzed using the ARGoS simulator. Since these results do not have a correspondent in the simulated results, I call them *predicted results* or simply *predictions*. With the first model, it is now possible to quickly find the expected number of accomplished tasks of a new experiment, that is to say, an experiment with different parameter values than those run with the ARGoS simulator. The first model is more robust to the parameter's value changes, so it is more suited for doing this kind of prediction.

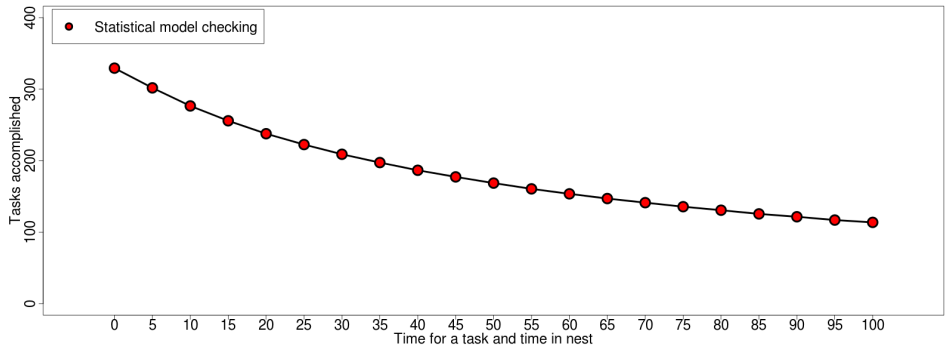
Figure 4.15 show us examples of predictions about the expected number of accomplished tasks within 1000s of various experiments. The first figure (4.15a) depicts



(a)



(b)



(c)

Figure 4.15: Predictions on the expected number of accomplished task over different scenarios. (a) Predictions on different environment sizes (the x-axis represent the number of robots, the area of the environment and the number of TAMs increases proportionally by following the formulae given in Section 4.4), the appearance rate is set at 0.01, the average task duration and time in nest are set at 5s. (b) Predictions on various appearance rates, the environment size is medium, the average task duration and time in nest are set at 5s. (c) Predictions on various average task durations and times in nest, both are set at the same value, the environment size is medium and the appearance rate is set at 0.01.

the expected number of accomplished tasks in function of the size of the system. In the figure, only the number of robots are shown on the x-axis but the area of the environment and the number of TAMs also increases proportionally to the number of robots (see formulae from Section 4.4). The appearance rate, the task duration and time in nest are fixed (appearance rate is set at 0.001, task duration and time in nest are set at 5s). We can see that the expected number of accomplished tasks follows a square root function depending on the number of robots. Interference does not play a major role in the result of the system, as the density of the robots in the arena is kept constant. The main reason why the number of accomplished tasks does not increase linearly in function of the number of robots is because the robots have to cover more distance to a task or the nest. The second figure (4.15b) depicts the expected number of accomplished tasks in function of the appearance rate. The size of the experiments is medium and the time in nest and task duration of those experiments are set at 5s. In the figure, we can see that when the appearance rate is at 0.01 the system performance does not improve further and when that the appearance rate is lower than 0.00001 almost no task is accomplished. This could be consider as a low limit of the flexibility of the system with respect to the task appearance. The third figure (4.15c) depicts the expected number of accomplished tasks in function of the average task duration and time in nest. The function first decreases relatively fast but afterwards decreases slower and slower. This is due to the fact that the searching time of the e-pucks becomes smaller. The e-pucks stay more time in the TAMs and therefore the other TAMs have more time to be available.

The experiments of the first figure took 2399s, those of the second figure took 317s and those of the last figure took 269s. Those times are really low compared to the time it would have taken with the ARGoS simulator, that is about 465s for each experiment.

4.4.4 Flexibility analysis

The results obtained using the ARGoS simulator and with model checking proved us that our foraging system is flexible. Indeed, the system gave good results with the different parameter values, that is to say, tasks were accomplished in all the various environments. The previous sections proved us that we were able to model the system and that with statistical model checking we can formally verify flexibility of the system.

Chapter 5

Conclusion

Flexibility is one of the most important properties of swarm robotic systems. Flexibility is important as swarm robotics systems are generally developed with the perspective of being able to work in various environments and not for a specific task. However, the available definitions of flexibility in literature are usually not precise and get often mixed together with the definitions of adaptability and scalability.

In swarm robotic, properties of a system are usually verified in an informal way, that is to say, by running multiple trials of the system on various environments and by analyzing the results. This method is time consuming and does not assure the correctness of the analysis. In fact, this method of multiple trials only take into account a small sample of all possible runs. A solution is to use model checking, which is able to formally model and analyze swarm robotic systems. After having model the system, model checking can verify all the possible outcomes of the executions of a system and therefore give complete results.

In this work, I started by giving a new definition of flexibility in swarm robotic systems. To do so, I had also to redefine properly adaptability in swarm robotic system in order to make the distinction clear. Flexibility is defined in this master as "the capacity of a system to achieve similarly performing collective behaviors in diverse environments without changing the behaviors of the individuals of the system". Adaptability became then the capacity of the system to generate a solution for an unforeseen problem. Giving a new definition of flexibility might seem of little use in practical the development and analysis of swarm robotic system. Therefore, I proposed a practical approach to the formal verification of flexibility using model checking. The approach was validated on a specific swarm robotic scenario: a foraging system.

First a simulated system of the foraging scenario was developed on ARGoS. Afterwards, two different models were developed. The first one was really close to the real system in terms of the individual robot's behavior. The problem of the first model was that the state space was too big to perform probabilistic model checking and thus only statistical model checking was possible. Consequently, I

tried to develop a smaller model. However, the second model had a different limit: it is necessary to retrieve information from the simulated system each time we want to verify how the system perform in a new environment.

The two models gave satisfying results. The expected number of accomplished tasks obtained by the two models were close to the average number of accomplished tasks obtained using the simulated system. The results of the simulated system and the statistical model checking on the models proved that the foraging system is flexible. The verification of the flexibility of the presented case study was an important confirmation of the validity of my method. This master thesis was concluded by using the first model for predictions about the number of accomplished tasks on environments using parameter values that were not simulated using ARGoS.

The next challenge would certainly be to apply the same approach using data from experiments with real robots. The fact that the model is representative for a simulated system do not ensure that it will be for the corresponding real-world system. Unexpected interference or problems with the TAM detection can occur.

In this master thesis the verification of flexibility has been done on only one specific system. As future work, the approach used in this work could be tested on different, more complex, case studies.

An interesting future work could be to verify the foraging case study in environments with different shapes and environment with obstacles. The models may no longer be accurate for those new environments and will maybe need some rectifications.

Finally, this approach for formally verifying flexibility could be used in the property-driven design approach. It could even a be a basic requirement of all the swarm robotic systems.

Appendix A

PRISM codes

A.1 Aggregation application

A.1.1 DTMC model of the aggregation application

code received from Brambilla et al. [8]

```
dtmc

const int N_tot;

global a: [0..N_tot] init 0;
global b: [0..N_tot] init 0;
global c: [0..N_tot] init N_tot;

const double Pca =0.0576; //0.0784;
const double Pac_max = 0.12;
const double Pbc_max = Pac_max;
formula Pcb = Pca;

// set 8 for small groups, 15 for 50 robots
formula Pac_sw = Pac_max*(1-(min(a,15)/15));
formula Pbc_sw = Pbc_max*(1-(min(b,15)/15));

module swarm
  [] (a>0) & (a<N_tot) & (b>0) & (b<N_tot) & (c>0) & (c<N_tot) //
     1
    -> Pca:(a'=a+1)&(c'=c-1) + Pcb:(b'=b+1)&(c'=c-1) + // go in an
       aggregate area
       Pac_sw:(a'=a-1)&(c'=c+1) + Pbc_sw:(b'=b-1)&(c'=c+1) + //
          leave an aggregate area
       (1-Pca-Pcb-Pac_sw-Pbc_sw):true;

  [] (a=0) & (b>0) & (b<N_tot) & (c>0) & (c<N_tot) // 2
    -> Pca:(a'=a+1)&(c'=c-1) + Pcb:(b'=b+1)&(c'=c-1) + // go in an
       aggregate area
```

```
        Pbc_sw:(b'=b-1)&(c'=c+1) + // leave an aggregate area
        (1-Pca-Pcb-Pbc_sw):true;

  [] (b=0) & (a>0) & (a<N_tot) & (c>0) & (c<N_tot) // 4
    -> Pca:(a'=a+1)&(c'=c-1) + Pcb:(b'=b+1)&(c'=c-1) + // go in an
        aggregate area
        Pac_sw:(a'=a-1)&(c'=c+1) + // leave an aggregate area
        (1-Pca-Pcb-Pac_sw):true;

  [] (c=0) & (a>0) & (a<N_tot) & (b>0) & (b<N_tot) // 6
    -> Pac_sw:(a'=a-1)&(c'=c+1) + Pbc_sw:(b'=b-1)&(c'=c+1) + //
        leave an aggregate area
        (1-Pac_sw-Pbc_sw):true;

  [] (a=0) & (b=0) & (c=N_tot) // 8
    -> Pca:(a'=a+1)&(c'=c-1) + Pcb:(b'=b+1)&(c'=c-1) + // go in an
        aggregate area
        (1-Pca-Pcb):true;

  [] (a=N_tot) & (b=0) & (c=0) // 9
    -> Pac_sw:(a'=a-1)&(c'=c+1) + // leave aggregate A
        (1-Pac_sw):true;

  [] (a=0) & (b=N_tot) & (c=0) // 10
    -> Pbc_sw:(b'=b-1)&(c'=c+1) + // leave aggregate A
        (1-Pbc_sw):true;
endmodule

label "aggregate" = (a=N_tot)|(b=N_tot);

rewards
  a=N_tot: 1;
  b=N_tot: 1;
endrewards
```

A.1.2 CTMC model of the aggregation application

```
ctmc

const int N_tot;


const double Pca = 0.0576;  //0.0784;
const double Pac_max = 0.12;
const double Pbc_max = Pac_max;

formula Pcb = Pca;

// set 8 for small groups, 15 for 50 robots
formula Pac_sw = max(Pac_max*(1-(min(a,15)/15)), 0.00001);
formula Pbc_sw = max(Pbc_max*(1-(min(b,15)/15)), 0.00001);
```

```
formula rate_Pcb = -log(1-Pcb, 2.71828183)/1;   // 1 sec
formula rate_Pca = -log(1-Pca, 2.71828183)/1;
formula rate_Pac = -log(1-Pac_sw, 2.71828183)/1;
formula rate_Pbc = -log(1-Pbc_sw, 2.71828183)/1;

module c_state
  c: [0..N_tot] init N_tot;

  [c_to_a]  (c>0) -> rate_Pca : (c' = c-1);
  [c_to_b]  (c>0) -> rate_Pcb : (c' = c-1);
  [a_to_c]  (c<N_tot) -> rate_Pac : (c' = c+1);
  [b_to_c]  (c<N_tot) -> rate_Pbc : (c' = c+1);
endmodule

module a_state
  a: [0..N_tot] init 0;

  [c_to_a] (a<N_tot) -> 1 : (a' = a+1);
  [a_to_c] (a>0) -> 1 : (a' = a-1);
endmodule

module b_state
  b: [0..N_tot] init 0;

  [c_to_b]   (b<N_tot) -> 1 : (b' = b+1);
  [b_to_c]   (b>0) -> 1 : (b' = b-1);
endmodule

label "aggregate" = (a=N_tot)|(b=N_tot);

rewards
  a=N_tot: 1;
  b=N_tot: 1;
endrewards
```

A.2 Foraging application

A.2.1 First model

```
ctmc

const int N_robots;
const int N_booths_side;
const double size;
const double Average_prey;
const double Average_nest;
const double Appearance;
```

```
formula N_nests = N_booths;
formula N_booths = 3*N_booths_side;
formula square_size = size*size;
formula max_cover = 1.5*size -0.63; // 2 * 0.5^2 + 0.13
formula max_cover_nest = 0.5*size -0.13;

const double k = 0.09;

// Finding probabilities
formula Ps_eb =  k*((0.13+max_cover*total_free_booths/N_booths)/
    square_size);
formula Ph_en = k*((0.13+max_cover_nest*total_free_nests/N_nests)/
    square_size);

// Rates of e-pucks behavior
formula rate_s_eb = -s*log(1-Ps_eb, 2.71828183);
formula rate_h_en = -h*log(1-Ph_en, 2.71828183);
formula rate_eb_r = (1-pow(2.71828183, -1/5))*eb; // 5 sec to
    enter booth
formula rate_r_h  = (1-pow(2.71828183, -1/Average_prey))*r;
formula rate_en_d = (1-pow(2.71828183, -1/5))*en; // 5 sec to
    enter nest
formula rate_d_s  = (1-pow(2.71828183, -1/Average_nest))*d;

// Rate of Task-TAM behavior
formula rate_booth_creation = Appearance*10*(N_booths-
    total_active_booths);

// Totals
formula total_in_booth = eb + r;
formula total_in_nest  = en + d;
formula total_free_booths = b - r;
formula total_free_nests = N_nests - d;
formula total_active_booths = b;

// States
module searching
  s: [0..N_robots] init N_robots;

  [s_to_eb] (s>0) & (total_in_booth < total_active_booths) ->
      rate_s_eb : (s' = s-1);
  [d_to_s] (s<N_robots) & (d > 0) -> 1: (s' = s+1);
endmodule

module enter_booth
  eb: [0..N_robots] init 0;

  [s_to_eb] (s>0) & (total_in_booth < total_active_booths) -> 1: (
      eb' = eb + 1);
  [eb_to_r] (eb>0) & (r < total_active_booths) -> rate_eb_r : (eb'
      = eb-1);
endmodule

module retrieve
```

```
  r: [0..N_booths] init 0;

  [eb_to_r] (eb>0) & (r < total_active_booths) -> 1: (r' = r + 1);
  [r_to_h] (r > 0) & (h < N_robots) -> rate_r_h : (r' = r - 1);
endmodule

module homing
  h: [0..N_robots] init 0;

  [r_to_h] (r > 0) & (h < N_robots) -> 1: (h' = h + 1);
  [h_to_en] (h > 0) & (total_in_nest < N_nests) -> rate_h_en : (h'
      = h - 1);
endmodule


module enter_nest
  en: [0..N_nests] init 0;

  [h_to_en] (h > 0) & (total_in_nest < N_nests) -> 1: (en' = en +
     1);
  [en_to_d] (d < N_nests) & (en > 0) -> rate_en_d : (en' = en - 1)
     ;
endmodule


module deposit
  d: [0..N_nests] init 0;

  [en_to_d] (d < N_nests) & (en > 0) -> 1: (d' = d + 1);
  [d_to_s] (s<N_robots) & (d > 0) -> rate_d_s : (d' = d - 1);
endmodule

module booths
  b: [0..N_booths] init 0;

  [] (b < N_booths) -> rate_booth_creation: (b' = b + 1);
  [r_to_h] (b > 0) -> 1: (b' = b - 1);
endmodule

rewards "num_preys"
  [d_to_s] true: 1;
endrewards
```

shell script for running the 36 experiments

```
#!/bin/bash
robot=(10 50 100)
booth=(4 20 40)
size=(2 4.5 6.3)

START=$(date +%s)
```

```
rm result_first_model.DAT
for i in 0 1 2
do
  for j in 5.0 50.0
  do
    for k in 5.0 50.0
    do
      for l in 0.01 0.001 0.0001
      do
      echo ${robot[$i]} ${booth[$i]} ${size[$i]} $j $k $l
      var=$(./prism ../../flexibility_modelling_model.pm ../../
          flexibility_modelling_properties.pctl -sim -const
          N_robots=${robot[$i]},N_booths_side=${booth[$i]},size=${
          size[$i]},Appearance=$l,Average_prey=$j,Average_nest=$k|
          grep Result)
      test=${var//[^0-9.]/}
      echo $test>>result_first_model.DAT
      done
    done
  done
done

END=$(date +%s)
DIFF=$(( $END - $START ))
echo "It took $DIFF seconds"
```

A.2.2 Second model

```
ctmc

// Parameters
const double av_time_to_TAM;
const double av_time_to_nest;
const double Average_prey;
const double Average_nest;
const int N_robots;
const int N_TAMs_side;

// Compute the number of TAMs
formula N_nests = N_TAMs;
formula N_TAMs = 3*N_TAMs_side;

// Rate of one e-puck
const double attb = (1-pow(2.71828183, -1/(av_time_to_TAM/10)));
const double attn = (1-pow(2.71828183, -1/(av_time_to_nest/10)));
const double ap = (1-pow(2.71828183, -1/Average_prey));
const double an = (1-pow(2.71828183, -1/Average_nest));

// Rate of the transitions
formula rate_s_r = s*attb;
formula rate_h_d = h*attn;
```

```
formula rate_r_h  = r*ap;
formula rate_d_s  = d*an;

// States
module searching
  s: [0..N_robots] init N_robots;

  [s_to_r] (s>0) & (r<N_TAMs) -> rate_s_r : (s' = s-1);
  [d_to_s] (s<N_robots) & (d > 0) -> 1: (s' = s+1);
endmodule

module retrieve
  r: [0..N_TAMs] init 0;

  [s_to_r] (s>0) & (r<N_TAMs) -> 1: (r' = r + 1);
  [r_to_h] (r > 0) & (h < N_robots) -> rate_r_h : (r' = r - 1);
endmodule

module homing
  h: [0..N_robots] init 0;

  [r_to_h] (r > 0) & (h < N_robots) -> 1: (h' = h + 1);
  [h_to_d] (h > 0) & (d < N_nests) -> rate_h_d : (h' = h - 1);
endmodule

module deposit
  d: [0..N_nests] init 0;

  [h_to_d] (d < N_nests) & (h > 0) -> 1: (d' = d + 1);
  [d_to_s] (s<N_robots) & (d > 0) -> rate_d_s : (d' = d - 1);
endmodule

rewards "num_preys"
  [d_to_s] true: 1;
endrewards
```

shell script for running the 36 experiments

```
#!/bin/bash
robot=(10 50 100)
booth=(4 20 40)
size=(2 4.5 6.3)

START=$(date +%s)

av_TAM=(221.1486  555.3168 6655.9511  261.4530  575.5689 6633.6402
      191.1158 330.8427 6367.3425  226.0973  363.0567 6462.3707
    426.5813  690.8654 6034.8633  492.5073  715.3099 6044.8289
    423.7321  593.3012 5798.4460 453.6033  623.4831 5762.0156
    574.0781  866.7040 5248.8036  650.4522 887.8802 5236.8196
    602.4643  787.5724 4996.1445  622.5276  813.6353 4991.6284)
av_nest=(428.6403  457.8077  432.8141  389.8783  413.0894
```

```
    420.5826 1009.4080 953.0026  479.7094  774.0712  757.2592
    471.2114 1106.8012 1349.7913 1538.6484 1146.0122 1280.5287
    1512.1887 1423.4302 1456.9150 1530.6720 1282.0523 1346.5566
    1490.3418 1522.7325 2034.0815 2553.6894 1707.3290 1975.3590
    2498.3243 1732.0653 1914.7340 2473.1632 1689.9505 1867.9578
    2408.0125)

counter=0
rm results.dat
for i in 0 1 2
do
  for j in 5.0 50.0
  do
    for k in 5.0 50.0
    do
      for l in 0.01 0.001 0.0001
      do
      echo ${robot[$i]} ${booth[$i]} ${size[$i]} $j $k $l
      var=$(./prism ../../flexibility_modelling_small_model.pm
          ../../flexibility_modelling_properties.pctl -sim -const
          N_robots=${robot[$i]},N_booths_side=${booth[$i]},
          av_time_to_booth=${av_TAM[$counter]},av_time_to_nest=${
          av_nest[$counter]},Appearance=$l,Average_prey=$j,
          Average_nest=$k|grep Result)
      test=${var//[^0-9.]/}
      echo $test>>results.dat
      counter=$counter+1
      done
    done
  done
done

END=$(date +%s)
DIFF=$(( $END - $START ))
echo "It took $DIFF seconds"
```

Bibliography

[1] E. Şahin. Swarm robotics: From sources of inspiration to domains of application. In *Swarm Robotics*, pages 10–20. Springer, 2005.

[2] A. Aziz, R. Brayton, S. Edwards, G. Hachtel, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. Ranjan, A. Sangiovanni-Vincentelli, S. Sarwary, T. Shiple, F. Somenzi, G. Swamy, and T. Villa. VIS User's Manual, 2012. URL http://embedded.eecs.berkeley.edu/research/vis/doc/VisUser/vis_user/vis_user.html.

[3] L. Bayindir and E. Sahin. A review of studies in swarm robotics. *Turkish Journal of Electrical Engineering*, 15:115–147, 2007.

[4] G. Beni. From swarm intelligence to swarm robotics. *Conference on Swarm Robotics*, pages 1–9, 2005.

[5] P. E. Black. Finite state machine. In *Dictionary of Algorithms and Data Structures*. U.S. National Institute of Standards and Technology, 2008.

[6] C. Blum and D. Merkle. *Swarm intelligence: introduction and applications*. Natural Computing Series. Springer, 2008.

[7] E. Bonabeau, M. Dorigo, and G. Theraulaz. *From Natural to Artificial Swarm Intelligence*. Oxford University Press, 1999.

[8] M. Brambilla, C. Pinciroli, M. Birattari, and M. Dorigo. Property-driven design for swarm robotics. In Conitzer, Winikoff, Padgham, and Van Der Hoek, editors, *AAMAS 2012 International Conference on Autonomous Agents and Multiagent Systems*. ACM Press, 2012.

[9] I. Bratko. *Prolog Programming for Artificial Intelligence*. Addison Wesley, 2001.

[10] A. Brutschy, Gi. Pini, N. Baiboun, A. Decugnière, and M. Birattari. The IRIDIA TAM: A device for task abstraction for the e-puck robot. Technical report, IRIDIA, Université Libre de Bruxelles, 2010.

[11] S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems*. Princeton Studies in Complexity. Princeton University Press, 2001.

[12] D. H. Chadwick. With a remarkable array of communication skills, weaver ants may have perfected social networking. *National Geographic*, 2011.

[13] E. M. Clarke, O. Grumberg, and D. A. Peled. Model Checking. *Journal of the American Statistical Association*, 962:314, 1999.

[14] M. Dorigo. Swarm Robotics : The Coordination of Robots via Swarm Intelligence Principles. *BiologicallyInspired Collaborative Computing*, 268, 2008.

[15] M. Dorigo and E. Şahin. Guest Editorial. *Autonomous Robots*, 17:111–113, 2004.

[16] M. Dorigo and T. Stützle. Ant Colony Optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39, 2004.

[17] M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, D. Burnier, A. Campo, A. L. Christensen, A. Decugnière, G. A. Di Caro, F. Ducatelle, E. Ferrante, A. Förster, J Martinez. Gonzales, J. Guzzi, V. Longchamp, S. Magnenat, N. Mathews, M. De Oca, R. O'Grady, C. Pinciroli, G. Pini, P. Rétornaz, J. Roberts, V. Sperati, T. Stirling, A. Stranieri, T. Stützle, V. Trianni, E. Tuci, A. E. Turgut, and F. Vaussard. Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics Automation Magazine*, 2011.

[18] M. Dowson. The Ariane 5 software failure. *ACM SIGSOFT Software Engineering Notes*, 22(2):84, 1997.

[19] L. Esch. *Mathématique pour économistes et gestionnaires*. Supérieur, De Boeck, 3 edition, 2006.

[20] A. Fehnker and P. Gao. Formal Verification and Simulation for Performance Analysis for Probabilistic Broadcast Protocols. In *Conference on Ad-Hoc, Mobile, and Wireless Networks*, pages 128–141, 2006.

[21] S. Garnier, J. Gautrais, and G. Theraulaz. The biological principles of swarm intelligence. *Swarm Intelligence*, 1:3–31, 2007.

[22] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535, 1994.

[23] J. Harrison. Formal Verification In Industry, 1999. URL http://www.cl.cam.ac.uk/~jrh13/slides/types-04sep99/slides1.pdf.

[24] S. Konur and C. Dixon. Formal verification of probabilistic swarm behaviours. In *Second International Conference on Software Engineering and Formal Methods*, 2011.

[25] S. Konur, C. Dixon, and M. Fisher. Analysing robot swarm behaviour via probabilistic model checking. *Robotics and Autonomous Systems*, 60:199–213, 2012.

[26] C. R. Kube and E. Bonabeau. Cooperative transport by ants and robots. *Robotics and Autonomous Systems*, 30(1-2):85–101, 2000.

[27] M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. PRISM 4.0: verification of probabilistic real-time systems. *24th International Conference on Distributed Computing Systems Workshops 2004 Proceedings*, 6806:585–591, 2004.

[28] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In M Bernardo and J Hillston, editors, *Formal Methods for Performance*, number 9 in LNCS (Tutorial Volume), pages 220–270. Springer, 2007.

[29] S. I. Landau. *Cambridge Dictionary of American English*. Cambridge University Press, 1999.

[30] K. Lerman, A. Martinoli, and A. Galstyan. A review of probabilistic macroscopic models for swarm robotic systems. *Lecture Notes in Computer Science*, 3342:143–152, 2004.

[31] W. Liu and A. Winfield. A macroscopic probabilistic model of adaptive foraging in swarm robotics systems. *International Journal of Robotics Research*, 29(14): 1743–1760, 2009.

[32] W. Liu and A. Winfield. Open-hardware e-puck Linux extension board for experimental swarm robotics research. *Microprocessors and Microsystems*, 35: 60–67, 2010.

[33] W. Liu, Al. Winfield, J. Sa, J. Chen, and L. Dou. Strategies for Energy Optimisation in a Swarm of Foraging Robots. *Proceedings of the 2nd international conference on Swarm robotics*, 4433:14–26, 2007.

[34] G. Metta, L. Natale, F. Nori, G. Sandini, D. Vernon, L. Fadiga, C. Von Hofsten, K. Rosander, M. Lopes, J. Santos-Victor, A. Bernardino, and L. Montesano. The iCub humanoid robot: an open-systems platform for research in cognitive development. *Neural Networks*, 23(8-9):1125–1134, 2010.

[35] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, J.-C. Zufferey, D. Floreano, and A. Martinoli. The e-puck, a Robot Designed for Education in Engineering. In *Conference on Autonomous Robot Systems and Competition*, number 1, pages 59–65. IPCB-Instituto Politécnico de Castelo Branco, 2009.

[36] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G.D. Caro, F. Ducatelle, T. Stirling, A. Gutierrez, L. Gambardella, and M. Dorigo. ARGoS: A modular, multi-engine simulator for heterogeneous swarm robotics. In I-Ming Chen, Alessandro De Luca, Chad Jenkins, Danica Kragic, Nikos Papanikolopoulos, Frank Park, Lynne Parker, Shigeki Sugano, and Frank Van Der Stappen, editors, *International Conference on Intelligent Robots and Systems*, number March, pages 5027–5034. IRIDIA,

CoDE, Université Libre de Bruxelles, 50 Avenue F. Roosevelt, CP 194/6, 1050, Belgium, IEEE, 2011. ISBN 9781612844558.

[37] P. Stone. *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. Intelligent Robotics and Autonomous Agents. MIT Press, 2000.

[38] R. W. Weyhrauch. Prolegomena to a Theory of Mechanized Formal Reasoning. *Artificial Intelligence*, 13:133–170, 1980.