Parameterised Model Checking of Probabilistic Multi-Agent Systems

By Edoardo Pirovano

Department of Computing

Imperial College London

May 2021

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing at Imperial College London.

ABSTRACT

Swarm robotics has been put forward as a method of addressing a number of scenarios where scalability and robustness are desired. In order to deploy robotic swarms in safety-critical situations, it is necessary to verify their behaviour. Model checking gives a possible approach to do this; however, with traditional model checking techniques only systems of a finite size can be considered. This presents an issue for swarm systems, where the number of participants in the system is not known at design-time and may be arbitrarily large. To overcome this, parameterised model checking (PMC) techniques have been developed which enable the verification of systems where the number of participants is not known until run-time. However, protocols followed by robotic swarms are often stochastic in nature, and this cannot be modelled with current PMC techniques. This is the gap that this thesis aims to overcome.

In particular, two parameterised semantics for reasoning about multi-agent systems are extended to incorporate probabilities. One of these semantics is synchronous, whilst the other is interleaved. Abstract models which overapproximate the systems being considered are constructed using counter abstraction techniques. These abstract models are used to develop parameterised verification procedures for a number of specification logics on both bounded and unbounded traces. The decision procedures presented are shown to be sound, and in some cases also complete. Further, the techniques are extended to allow modelling of situations where agents may exhibit faulty behaviour, as well as scenarios where the strategic capabilities of the participants needs to be verified.

The procedures are all implemented in a novel verification toolkit called `PSV` (**P**robabilistic **S**warm **V**erifier), built on top of the probabilistic model checker PRISM. This toolkit is used to verify three case studies from both swarm robotics and other application domains.

ACKNOWLEDGMENTS

Firstly, I would like to thank my supervisor Alessio Lomuscio for his guidance. Without his valuable insights into my work and impeccable feedback, this thesis would never have been written. I would also like to thank all my colleagues, and in particular Michael Akintunde and Panagiotis Kouvaros, for many useful discussions. I am very grateful to my friend Benjamin Walker for a number of helpful comments on early drafts of my work. Finally, a massive thank you to my parents for their motivation and support throughout my PhD.

DECLARATION OF ORIGINALITY

I hereby declare that this thesis and the research presented within it are my own work, except where otherwise indicated. The work presented here has not been submitted for any other degree or professional qualification.

Edoardo Pirovano

COPYRIGHT

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter One

Introduction

Multi-agent systems (MAS) arise in a number of useful real-life scenarios, such as auctions and Internet of Things applications. In this thesis, however, most of the scenarios we consider will come from *swarm robotics*. Swarm robotics is an approach to performing a task by using a large number of robots that coordinate to accomplish their common goal. Robotic swarms have been put forward a good alternative to single robots in a number of scenarios [Şahin and Winfield, 2008], including tasks such as surveillance and maintenance of industrial plants.

There are several properties of swarm systems that make them a better solution than a single robot in many situations. One of these is *scalability* – the ability to operate with different group sizes [Şahin, 2005]. Scalability allows for a different number of robots to be deployed depending on the circumstances. For instance, one proposed application of swarm robotics is the automated monitoring of the condition of pipelines [Parrott et al., 2020]. In this application, the size of the swarm could be adapted depending on the length of the pipeline and the desired frequency of monitoring.

Another key property of swarm systems is *robustness* – the ability to continue to operate (usually at reduced functionality) despite failures in the individuals, or disturbances in the environment. Some proposed applications of swarm systems depend entirely on robustness. For instance, swarms have been put forward as a way of providing a communication network in a battlefield [Şahin, 2005], where agents would re-arrange themselves if one is destroyed

to keep the overall communication network intact. Even for applications where robustness is not part of the specification, it is still usually a desirable property.

In safety-critical situations like the ones considered above, it is necessary to verify that these swarm systems will behave as expected regardless of how many agents are present in the system. One technique that has been put forward to verify the behaviour of complex systems is that of model checking [Clarke et al., 1999]. Traditional model checking techniques can only be used to verify systems of a finite size and, therefore, are not suited to verifying properties of robotic swarms due to their unbounded nature. However, parameterised model checking [Bloem et al., 2015] extends traditional model checking to enable the verification of arbitrarily large systems.

A number of swarm protocols are stochastic in nature [Bonabeau et al., 1999; de Oca et al., 2011]. Existing parameterised model checking techniques mostly do not consider probabilities (with a couple of exceptions that are discussed in Section 8.2). Thus, with current techniques it is not possible to verify probabilistic specifications in multi-agent systems such as swarms that may be unbounded in size.

The overall aim of this thesis is to address this gap in current model checking techniques by developing a method to verify multi-agents systems that are both stochastic and possibly unbounded in size.

1.1 Objectives

This thesis will have three main objectives, corresponding to different stages in the research being carried out.

1. **Develop a semantics for reasoning about unbounded probabilistic MAS.** This objective will involve two parts. First, we will aim to define models for unbounded probabilistic multi-agent systems (UPMAS). To achieve this we will extend existing models for non-probabilistic MAS [Kouvaros and Lomuscio, 2016] to incorporate prob-

abilities. Secondly, we will define a number of logics for reasoning about properties of these UPMAS. To achieve this we will adapt existing logics from probabilistic model checking [Hansson and Jonsson, 1994] to allow us to reason about UPMAS.

2. **Develop decision procedures for our verification problems.** Having defined decision problems for the verification of UPMAS, we will aim to develop procedures that solve them. To do this, we will develop abstract models based on counter abstraction [Pnueli et al., 2002] that overapproximate the behaviour of arbitrarily large systems. As is typical in parameterised model checking, some of the decision problems we consider will be undecidable [Apt and Kozen, 1986]. In these cases, we will nonetheless aim to obtain useful partial decision procedures. We will also formally prove the soundness of all our decision procedures.

3. **Implement and assess the methods we have developed.** Having defined and proved the corectness of our decision procedures, we will implement them on top of the existing probabilistic model checker PRISM [Kwiatkowska et al., 2011]. Further, we will assess their usability by considering a number of case studies from swarm robotics and other application domains.

1.2 Contributions

This thesis makes contributions in a number of ways.

1. Conceptual

   - We introduce *synchronous probabilistic multi-agent systems* (SPMAS), a semantics for reasoning about synchronous multi-agent systems that are both probabilistic and possibly unbounded in size.

- We introduce *asynchronous probabilistic multi-agent systems* (APMAS), a semantics for reasoning about asynchronous multi-agent systems that are both probabilistic and possibly unbounded in size.
- We present the logics PLTL, PLTL_{k} and P[ATL*] which are closely based on existing logics but adapted to allow us to reason about probabilistic MAS.
- We introduce a framework for reasoning about faults in probabilistic multi-agent systems.

2. Theoretical

- We introduce a procedure for checking SPMAS against PLTL_{k} and prove that it is sound and complete.
- We introduce a procedure for checking APMAS against PLTL and prove that it is sound.
- We introduce a procedure for checking SPMAS against P[ATL*] and prove that is sound.
- We introduce a procedure for injecting faults into an APMAS to obtain a faulty system that can be verified using the above procedure.

3. Practical

- We introduce **P**robabilistic **S**warm **V**erifier (`PSV`), a toolkit implementing the procedures described above.
- We present implementations of three case studies in `PSV` and give results obtained for these case studies.

1.3 Publications

The results presented in this thesis have previously appeared, in a shorter form, in the following papers [Lomuscio and Pirovano, 2018, 2019, 2020a,b]:

- A. Lomuscio, E. Pirovano. Verifying Emergence of Bounded Time Properties in Probabilistic Swarm Systems. Proceedings of the 27th International Joint Conference on Artificial Intelligence and 23rd European Conference on Artificial Intelligence (IJCAI-ECAI18). Stockholm, Sweden. AAAI Press.

- A. Lomuscio, E. Pirovano. A Counter Abstraction Technique for the Verification of Probabilistic Swarm Systems. Proceedings of the 18th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS19). Montreal, Canada. IFAAMAS Press.

- A. Lomuscio, E. Pirovano. Parameterised Verification of Strategic Properties in Probabilistic Multi-Agent Systems. Proceedings of the 19th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS20). Auckland, New Zealand. IFAAMAS Press.

- A. Lomuscio, E. Pirovano. Verifying Fault-Tolerance in Probabilistic Swarm Systems. Proceedings of the 29th International Joint Conference on Artificial Intelligence and the 17th Pacific Rim International Conference on Artificial Intelligence (IJCAI-PRICAI20). Yokohama, Japan. AAAI Press.

This thesis builds on the papers above by combining the theoretical results into a unified presentation, and giving full proofs rather than the proof outlines included in our papers. Further, the individual tools presented in the papers are combined into one toolkit that supports all the different systems previously considered.

The definitions from each paper are pulled out into Chapter 3. The main theoretical results are then presented in the same order as they are listed above in Chapter 4, Chapter 5,

Section 6.1, and Section 6.2 respectively. Finally, the experimental results of the papers are again presented jointly in Chapter 7.

1.4 Thesis Outline

After introducing some of the notation that will be used in this thesis in the next section, the rest of the thesis is structured as follows.

In Chapter 2 we give a review of existing literature, both in probabilistic model checking and in parameterised model checking for multi-agent systems.

We introduce the theoretical background for this thesis in Chapter 3. In particular, we define our two frameworks for reasoning about unbounded probabilistic multi-agent systems (SPMAS and APMAS), and our specification logics for reasoning about their properties which are based on existing logics but adapted to our specific needs.

Our main theoretical contribution is in Chapters 4 and 5. In Chapter 4 we give our procedure for verifying SPMAS against PLTL_{k}, and prove that is sound and complete. Similarly, in Chapter 5 we give our procedure for verifying APMAS against PLTL, and prove that is sound.

In Chapter 6 we give two extensions of our results. In particular, we give a procedure for checking SPMAS against $\text{P}[\text{ATL}^*]$ (a richer logic that can describe strategic properties of what the agents can achieve) and a procedure for creating a faulty APMAS from a non-faulty one and a description of faults that may occur.

We present our toolkit `PSV`, which implements all the procedures described earlier, in Chapter 7. In the same chapter, we also introduce three case studies and give verification results for all of them obtained using `PSV`.

Finally, we conclude in Chapter 8 where we summarise the contributions we have made and compare this thesis to related work in areas outside of model checking. We also highlight some possible areas for future work in this closing chapter.

1.5 Notation

Before continuing with the rest of this thesis, we introduce a few non-standard pieces of notation that the reader may not have come across before. In particular:

- $\mathbb{N}_1$ is used to denote the set of all positive integers, i.e. $\mathbb{N}_1 = \mathbb{N} \setminus \{0\} = \{1, 2, 3, \ldots\}$.
- Given a positive integer $k \in \mathbb{N}_1$, we use $\dot{k}$ to denote the set of all positive integers up to and including k, i.e. $\dot{k} = \{1, \ldots, k\}$.
- Given a finite set X, we use $\mathcal{P}(X)$ to denote the power set of X sometimes also written as 2^X, i.e.
$$\mathcal{P}(X) = 2^X = \{A : A \subseteq X\}$$
- $\bar{1}_k \in \mathbb{N}_1^k$ is the vector of length k composed entirely of ones, i.e. $\bar{1}_k = (1, \ldots, 1)$. When it is clear from context, we omit the k and just write $\bar{1}$.

Chapter Two

Background

In this chapter, we summarise the main existing research in both probabilistic model checking and parameterised model checking. We conclude by discussing why neither of these is adequate for the verification of probabilistic swarm systems. This chapter aims to provide a background in the work that this thesis will build upon in order to enable the reader to understand the work presented and its novelty. We defer discussing less closely related approaches and comparing to these until Section 8.2.

2.1 Probabilistic Model Checking

Probabilistic model checking is a large and well-studied field, with many different models being studied. In this section we will discuss discrete-time Markov chains (DTMCs) and Markov decision processes (MDPs), and their use in the verification of multi-agent systems, as these aspects are most relevant to the rest of the thesis. Other models exist (such as continuous-time Markov chains [Kwiatkowska et al., 2007] and probabilistic timed automata [Norman et al., 2013]) and applications exist in many other domains (such as systems biology [Kwiatkowska et al., 2010] and network protocols [Duflot et al., 2010]). We do not cover these as they are not used in this thesis, and refer the reader to [Katoen, 2016] for a more comprehensive survey. Efficient implementations of all the techniques discussed

here exist in toolkits such as PRISM [Kwiatkowska et al., 2011] and STORM [Dehnert et al., 2017].

2.1.1 Probability Distributions and Measures

We now cover some background definitions on probability. We will assume some familiarity with the basics of probability theory, as can be found in a number of text books [Feller, 1968], though we briefly summarise the important definitions below. Our first definition is that of a discrete probability distribution, which assigns to each element of a set a probability.

Definition 2.1 (Discrete Probability Distribution)**.** *A discrete probability distribution over a countable set S is a function $\mu : S \to [0, 1]$ satisfying $\sum_{s \in S} \mu(s) = 1$.*

We use $Dist(S)$ to denote the set of all discrete probability distributions over S. We now go on to define the concept of a probability space.

Definition 2.2 (Probability Space)**.** *A probability space over a sample space Ω is a triple $(\Omega, \mathcal{F}, Pr)$, where $\mathcal{F} \subseteq \mathcal{P}(\Omega)$ is a σ-algebra over Σ, i.e. satisfies:*

1. *$\emptyset, \Omega \in \mathcal{F}$*
2. *if $A \in \mathcal{F}$, then $\Omega \setminus A \in \mathcal{F}$*
3. *if $A_i \in \mathcal{F}$ for all $i \in \mathbb{N}$ then $\cup_{i \in \mathbb{N}} A_i \in \mathcal{F}$*

and $Pr : \mathcal{F} \to [0, 1]$ is a probability measure over $(\Sigma, \mathcal{F})$, i.e. satisfies:

1. *$Pr(\emptyset) = 0$*
2. *$Pr(\Sigma) = 1$*
3. *For all countable pairwise disjoint sequences $A_1, A_2, \ldots$ of elements from $\mathcal{F}$ it is the case that $Pr(\cup_{i \in \mathbb{N}} A_i) = \sum_{i \in \mathbb{N}} Pr(A_i)$*

When working with a probability space $(\Omega, \mathcal{F}, Pr)$, we say that sets contained in $\mathcal{F}$ are *measurable*.

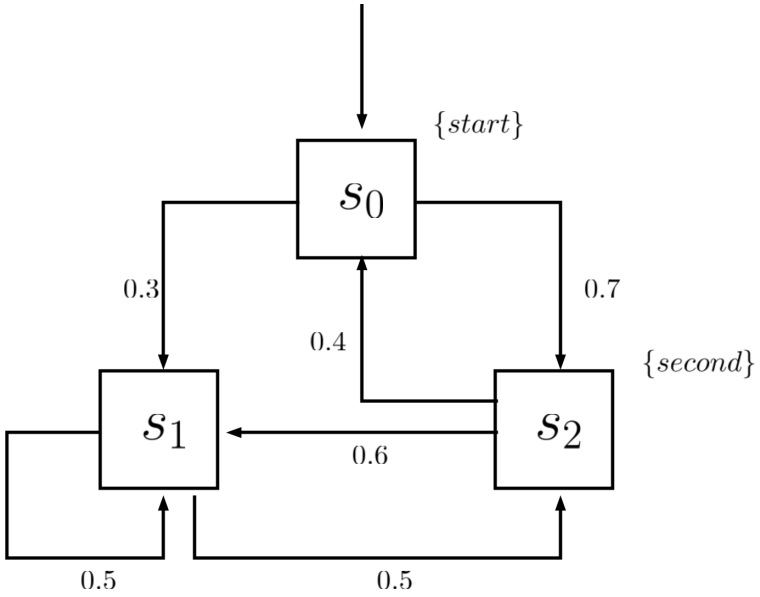


Figure 2.1 An example DTMC.

2.1.2 Discrete Time Markov Chains

We briefly summarise *discrete time Markov chains* (DTMCs). For more background on DTMCs and their model checking see [Kwiatkowska et al., 2007; Baier and Katoen, 2008; Kemeny et al., 1976].

Definition 2.3 (DTMC)**.** *A discrete time Markov chain (DTMC) is a tuple* $\mathcal{D} = \langle S, \iota, t, L \rangle$ *where* S *is a set of states,* $\iota \in S$ *is a distinguished initial state,* $t : S \to Dist(S)$ *is a transition function that returns a probability distribution on the next state given the current one and* $L : S \to \mathcal{P}(AP)$ *is a labelling function on a set* AP *of atomic propositions.*

For convenience, we will typically write $t(s_2 \mid s_1)$ as shorthand for $(t(s_1))(s_2)$, i.e. the probability of transitioning to s_2 from s_1. We use this notation whenever we consider a function that emits a probability distribution.

A *path* in a DTMC is a sequence of states $s_0 s_1 s_2 \ldots$ such that for every $i \in \mathbb{N}$ it is the case that $t(s_{i+1} \mid s_i) > 0$. We use $FPath_{\mathcal{D},s}$ and $IPath_{\mathcal{D},s}$ respectively, to denote the set of all finite and infinite paths in $\mathcal{D}$ starting from a state s. For a finite path we define its probability by $\mathbf{P}_{\mathcal{D}}(s_0 \ldots s_n) \triangleq \prod_{i=0}^{n-1} t(s_{i+1} \mid s_i)$. Following [Kemeny et al., 1976], this can be

extended to a probability space on the set of all infinite paths. In particular, we can define for each finite path its basic cylinder as follows.

Definition 2.4 (Basic Cylinder)**.** *Let $\rho \in FPath_{\mathcal{D},s}$, then we define its basic cylinder:*

$$C_\rho \triangleq \{\rho' \in IPath_{\mathcal{D},s} \mid \rho \in Prefs(\rho')\}$$

where $Prefs(s_0 s_1 s_2 \ldots) = \{s_0,\ s_0 s_1,\ s_0 s_1 s_2,\ \ldots\}$ is the set of finite prefixes of an infinite path.

Informally, the basic cylinder of a finite path is the set of infinite paths which are an extension of it. We can then use this to define the probability space $(IPath_{\mathcal{D},s}, \mathcal{F}_{\mathcal{D},s}, Pr_{\mathcal{D},s})$ where $\mathcal{F}_{\mathcal{D},s}$ is the smallest σ-algebra generated by the basic cylinders $\{C_\rho \mid \rho \in FPath_{\mathcal{D},s}\}$ and $Pr_{\mathcal{D},s}$ is the unique measure such that $Pr_{\mathcal{D},s}(C_\rho) = \mathbf{P}_{\mathcal{D}}(\rho)$ for all $\rho \in FPath_{\mathcal{D},s}$.

Example 2.1. *An example DTMC $\mathcal{D} = (S, \iota, t, L)$ can be seen in Figure 2.1. Here, $S = \{s_0, s_1, s_2\}$ and $\iota = s_0$. The transition probability is given as in the diagram. $L(s_0) = \{start\}$, $L(s_1) = \emptyset$, and $L(s_2) = \{second\}$. Some properties that we could consider include:*

- $Pr_{\mathcal{D},s_0}(\{\pi : \pi \text{ starts } s_0 s_1 s_2 s_0\}) = 0.3 \cdot 0.5 \cdot 0.4 = 0.06$
- $$\begin{aligned} Pr_{\mathcal{D},s_0}(\{(s_0 s_1 s_2)^\omega\}) &= \lim_{n\to\infty} Pr_{\mathcal{D},s_0}(\{\pi : \pi \text{ starts } (s_0 s_1 s_2)^n\}) \\ &= \lim_{n\to\infty} 0.3 \cdot 0.5 \cdot (0.4 \cdot 0.3 \cdot 0.5)^{n-1} \\ &= 0 \end{aligned}$$

We note that while all infinite paths in our example DTMC have probability 0, we can still assign meaningful probabilities to sets of paths that we are interested in if they share the same initial prefix.

DTMCs are typically checked against logics such as PCTL [Kwiatkowska et al., 2007]. One proposed algorithm for checking PCTL properties on DTMCs [Courcoubetis and Yannakakis, 1995] works in a similar way to the model checking algorithm for CTL [Clarke et al., 1986] by evaluating sets of states that satisfy sub-formulas and inductively building up to

the formula being checked. This gives an algorithm that is linear in the size of the formula, and polynomial in the number of states in the DTMC.

2.1.3 Markov Decision Processes

We now summarise some key aspects of *Markov decision processes* (MDPs) [Puterman, 1994]. We refer to [Baier and Katoen, 2008; Forejt et al., 2011] for more details. We mostly follow the notation used in [Forejt et al., 2011].

Definition 2.5 (MDP)**.** *A Markov decision process (MDP) is a tuple* $\mathcal{M} = \langle S, \iota, A, P, t, L\rangle$ *where* S *is a set of states,* $\iota \in S$ *is a distinguished initial state,* A *is a finite set of actions,* $P : S \to \mathcal{P}(A)$ *is a protocol function (such that* $P(s) \neq \emptyset$ *for all* $s \in S$*),* $t : S \times A \to Dist(S)$ *is a transition function and* $L : S \to \mathcal{P}(AP)$ *is a labelling function on a set* AP *of atomic propositions.*

Intuitively, a transition from a state s of an MDP occurs by first non-deterministically selecting some action $a \in P(s)$ and then transitioning to a new state according to the probability distribution given by $t(s, a)$. MDPs thus give a way of describing systems that include both probabilistic and non-deterministic choice, unlike DTMCs which do not capture the latter.

A *path* in an MDP is a sequence of states and actions $s_0 a_0 s_1 a_1 s_2 \ldots$ such that for all $i \in \mathbb{N}$ it is the case that $a_i \in P(s_i)$ and $t(s_{i+1} \mid s_i, a_i) > 0$. We use $FPath_{\mathcal{M}}$ ($IPath_{\mathcal{M}}$, respectively) to denote the set of all finite (infinite, respectively) paths starting from the initial state ι. For a finite path $\rho = s_0 a_0 \ldots s_n$, we use $last(\rho) \triangleq s_n$ to denote its last state.

In order to reason about the probability of a path occurring in an MDPs, we need a way to resolve the inherent non-determinism. This is captured by a strategy (also referred to as an *adversary*, *scheduler* or *policy* in some literature).

Definition 2.6 (Strategy)**.** *Given an MDP* $\mathcal{M} = \langle S, \iota, A, P, t, L\rangle$ *a strategy for* $\mathcal{M}$ *is a function* $\sigma : FPath_{\mathcal{M}} \to Dist(A)$ *such that for any finite path* $\rho \in FPath_{\mathcal{M}}$*, we have*

$\sigma(a \mid \rho) > 0$ *only if* $a \in P(last(\rho))$.

We denote by $Strat_{\mathcal{M}}$ the set of all strategies for $\mathcal{M}$. Various classes of strategies may be defined [Forejt et al., 2011]. Note that when maximising or minimising the probability of reaching a target set of states, it is sufficient to consider strategies that are memoryless (only make choices based on the final state of the path) and deterministic (only assign values from $\{0, 1\}$ to all actions). When it simplifies the presentation of our results we will consider such subclasses of strategies; for memoryless strategies we will consider a single state S instead of a path $FPath_{\mathcal{M}}$, and for deterministic ones we will return a single action from A instead of a distribution $Dist(A)$.

We now proceed to define the DTMC induced by a strategy on an MDP. Intuitively, this describes the purely probabilistic system that results from fixing a given choice of strategy in an MDP.

Definition 2.7 (Induced DTMC)**.** *Given an MDP* $\mathcal{M} = \langle S, \iota, A, P, t, L\rangle$ *and a strategy* $\sigma : FPath_{\mathcal{M}} \to Dist(A)$*, the induced DTMC is given by* $\mathcal{M}_\sigma = \langle FPath_{\mathcal{M}}, \iota, t', L'\rangle$ *where* $t' : FPath_{\mathcal{M}} \to Dist(FPath_{\mathcal{M}})$ *is given by:*

$$t'(\rho' \mid \rho) \triangleq \begin{cases} \sigma(a \mid \rho) \cdot t(s \mid last(\rho), a) & \text{if } \rho' = \rho as \\ 0 & \text{otherwise} \end{cases}$$

And $L' : FPath_{\mathcal{M}} \to \mathcal{P}(AP)$ *is given by* $L'(\rho) \triangleq L(last(\rho))$.

Notice that when considering memoryless strategies, instead of considering finite paths $FPath_{\mathcal{M}}$ in the induced DTMC, we can simply consider states S. We now go on to give an example of an MDP, a strategy, and the DTMC that this induces.

Example 2.2. *An example MDP* $\mathcal{M} = \langle S, \iota, A, P, t, L\rangle$ *can be seen in Figure 2.2. Here,* $S = \{s_0, s_1, s_2\}$ *and* $\iota = s_0$*. We have* $A = \{go, stay, tryMove, move\}$ *with* $P(s_0) = \{go\}$*,* $P(s_1) = \{stay, tryMove\}$*,* $P(s_2) = \{move\}$*. The transition probability is given as in the diagram.* $L(s_0) = \{start\}$*,* $L(s_1) = \emptyset$*, and* $L(s_2) = \{second\}$.

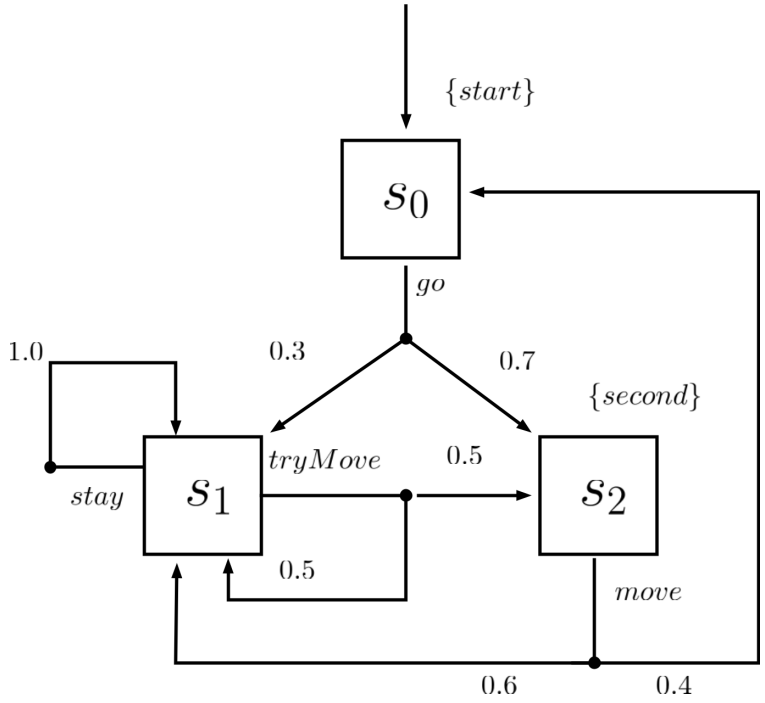


Figure 2.2 An example MDP.

One possible memoryless and deterministic strategy $\sigma : S \to A$ for this MDP could be given by $\sigma(s_0) = go$, $\sigma(s_1) = tryMove$, $\sigma(s_2) = move$. The induced DTMC M_σ that results from fixing this choice of strategy in the MDP is precisely the one from Example 2.1.

MDPs can also be verified against PCTL [Forejt et al., 2011]. As with DTMCs, a model checking algorithm [Bianco and de Alfaro, 1995] has been developed that is similar to the one for CTL [Clarke et al., 1986]. Once again, this algorithm is polynomial in the size of the model and linear in the size of the formula.

2.1.4 Applications to Multi-Agent Systems

There has been much work in using probabilistic model checking to verify swarm systems of a fixed size. For example, in [Gainer et al., 2016] a swarm protocol that aims to have drones search for a target is analysed by probabilistic model checking. There, the semantics for the scenario under consideration are expressed as DTMCs and properties in PCTL are checked against these DTMCs. In a similar fashion, in [Konur et al., 2012] a foraging protocol in which

drones retrieve food and bring it back to a nest is analysed by expressing it as a DTMC and verifying PCTL properties against it. In [Winfield et al., 2008], probabilistic models are constructed to reason about the behaviour of a swarm that is following an aggregation protocol (that is, the robots in the swarm have the goal of grouping together and remaining in a group).

In all these papers, counting abstractions are used which exploit the fact that all participants are behaviourally identical by not tracking the state of each agent but instead tracking only the total number of agents in each state. This reduces the state space explosion, but still limits the verification techniques to systems of a finite size. This thesis, on the other hand, aims to address systems in which there may be an unbounded number of participants at run-time.

In [Lukina et al., 2018], statistical model checking is used to analyse a flocking protocol. Statistical model checking is a technique used when analysing probabilistic systems that have large or infinite state spaces. In such systems, finding the exact probability of an event occurring may be impossible or prohibitively expensive. Thus, the system is instead simulated and a number of runs are carried out in order to estimate the probability to within acceptable statistical guarantees.

One recent paper [Gainer et al., 2018] has addressed the problem of efficiently verifying a number of closely related Markov chains by exploiting the similarity between them. This work may prove useful in developing verification techniques for drone swarms. However, in this thesis we intend to take a different angle and instead aim to develop techniques that will allow checking of properties for *all* instances of a system, as has been done for some classes of non-probabilistic systems by work on parametric model checking (discussed in Section 2.2).

There has been little work in the verification of probabilistic systems with a potentially unbounded number of agents. The few pieces of work that have been carried out [Graham, 2008; Fournier, 2015] used semantics and case studies focussed on network protocols. It is not clear if it is possible to adapt this work to model swarm algorithms, as we do in this

thesis.

2.2 Parameterised Model Checking

In many scenarios it is not sufficient to check properties on systems with a fixed number of agents; it is necessary for specifications to hold in instances of *any* size, since the number of agents in a system may not be known at design time. This is the aim of parameterised model checking.

Given a template description of a system that for any given size returns a concrete instance of that size, the parameterised model checking problem (PMCP) involves checking whether a certain specification holds in all (infinitely many) instances of the system [Bloem et al., 2015]. This can be formalised as:

Definition 2.8 (PMCP)**.** *Let $\mathcal{S}$ be a template description of a system, which when provided with an integer n giving the number of agents in a particular instantiation of the system returns this concrete system. Let ϕ be a formula in a suitable specification logic. Then, the PMCP is concerned with checking if:*

$$\forall n \in \mathbb{N}_1 : \mathcal{S}(n) \models \phi$$

Depending on the context, the details of the template and the specification logic can be adapted. We discuss one semantics that is particularly useful for model checking of swarm systems later in this section. In its most general formulation, the problem is known to be undecidable [Apt and Kozen, 1986]. Nonetheless, given the problem's importance, it is desirable to identify decidable fragments of it and develop and implement algorithms to solve these fragments. This has been a very active area of research, which we give a survey of in this section.

Closely related to the work on parametric model checking is a line of work that considers fault tolerance in parametric systems. We defer discussing this till Section 2.3.

Early work in parameterised model checking focussed on networks of communicating processes [Clarke et al., 1989]. These methods relied on restricting the communication pattern between the processes, for example by arranging the processes in a ring and having each process only communicate with its left and right neighbour, to restore decidability. These restrictions, while useful to model certain network protocols, prove too limiting when considering drone swarms where richer communication patterns are needed.

One notion that is often used to restore decidability in certain fragments of the PMCP is that of a *cut-off* [Kaiser et al., 2010; Kouvaros and Lomuscio, 2013; Spalazzi and Spegni, 2020]. Informally, a cut-off for a given property is a threshold such that, for any system larger than the cut-off, the property will hold iff it holds for the cut-off system. More formally:

Definition 2.9 (Cut-off)**.** *Let $\mathcal{S}$ be a parameterised system, and ϕ a formula. Then, $c \in \mathbb{N}_1$ is a cut-off for ϕ in $\mathcal{S}$ iff $\mathcal{S}(c) \models \phi$ implies $\mathcal{S}(n) \models \phi$ for all $n \geq c$.*

Notice that a cut-off identification procedure gives a decision procedure for the PMCP by identifying a cut-off c and then checking all systems of size up to c. Thus, such a procedure cannot exist in general. Indeed, there are systems and formulas for which no cut-off exists, an example of which can be found in [Kouvaros and Lomuscio, 2016]. Nonetheless such procedures exist for certain classes of systems and can be used to verify meaningful properties.

Full details and correctness proofs for a number of cut-off identification procedures can be found in [Kouvaros, 2015]. These have been implemented in a toolkit on top of `MCMAS` [Lomuscio et al., 2017] known as `MCMAS-P` [MCMAS-P, 2015]. We briefly summarise the results here. The semantics considered are referred to as parameterised interleaved interpreted systems (PIISs). These are an extension of IIS [Fagin et al., 1995] to deal with systems of unbounded size. They involve template agents (one for each type of agent in the system) which give a description of the behaviour of an individual agent, and an environment which captures the rest of the behaviour of the system.

An agent template $T = \langle L, \iota, Act, P, t \rangle$ contains a finite, non-empty set of local agent states L together with a distinguished initial state $\iota \in L$. The non-empty set $Act = A \cup AE \cup GS$ gives the actions that can be performed by the agents, which can be of a number of (disjoint) different types. The actions are performed in compliance with a protocol $P : L \to \mathcal{P}(Act)$ that defines which actions are enabled in each state. The agent's transition function $t : L \times Act \to L$ describes the evolution of the agent's state: given its local state and the action performed, it returns its new local state.

Each type of action involves different sets of agents that participate and update their state accordingly. This is formalised in Definition 2.11. Informally, the types of action are:

- *Asynchronous (A):* Performed independently by one agent.
- *Agent-environment (AE):* Performed synchronously by one agent and the environment.
- *Global-synchronous (GS):* Performed synchronously by all agents in the system and the environment.

In [Kouvaros and Lomuscio, 2016] two further types of actions (role-synchronous, and multi-role) are also defined. As these are not used in this thesis, we do not discuss these.

The environment $e = \langle L_e, \iota_e, Act_e, P_e, t_e \rangle$ defines a finite, non-empty set of local states L_E, a distinguished initial state $\iota_E \in L_E$, a non-empty set of actions $Act_E = A_E \cup AE \cup GS$, a protocol $P_E : L_E \to \mathcal{P}(Act_E)$, and a transition function $t_E : L_E \times Act_E \to L_E$.

Definition 2.10 (PIIS)**.** *A parameterised interleaved interpreted system (PIIS) is a tuple $\mathcal{S} = \langle T, e, \mathcal{V} \rangle$, where $\mathcal{V} : L \to 2^{AP}$ is a labelling function on the agent template's states for a set AP of atomic propositions.*

Each PIIS describes an unbounded collection of concrete systems obtained by choosing different numbers of agents in the system. Given a PIIS $\mathcal{S}$ and $n \in \mathbb{N}_1$ the IIS $\mathcal{S}(n)$ of n agents is the result of the composition of n copies of T with the environment. Denote the set of concrete agents instantiated from T by $\mathcal{A} = \{1, \ldots, n\}$. A *global state* $g = \langle l_1, \ldots, l_n, l_E \rangle$

is a tuple of local states for all the agents and the environment in $\mathcal{S}(n)$; it describes the system at a particular instant of time. We denote the set of all such global states by G. For a global state g, we let $g.i$ denote the local state of agent i in g and $g.e$ denote the state of the environment in g. The system's global states evolve over time in compliance with the global transition relation.

Definition 2.11 (Global Transition Relation)**.** *The global transition relation* $R \subseteq G \times (Act \cup Act_E) \times G$ *is defined as* $(g, a, g') \in R$ *iff one of the following holds:*

- (Asynchronous). *(i)* $a \in A \cup A_E$*; (ii) there is* $i \in \mathcal{A} \cup \{E\}$ *s.t.* $a \in P(g.i)$ *and* $t(g.i, a) = g'.i$*; (iii) for all* $j \neq i$*,* $g.j = g'.j$*.*
- (Agent-environment). *(i)* $a \in AE$*; (ii) there is* $i \in \mathcal{A}$ *s.t.* $a \in P(g.i)$ *and* $t(g.i, a) = g'.i$*; (iii)* $a \in P_E(g.e)$ *and* $t_E(g.e, a) = g'.e$*; (iv) for all* $j \neq i$*,* $j \neq e$*,* $g.j = g'.j$*.*
- (Global-synchronous). *(i)* $a \in GS$*; (ii) for all* $i \in \mathcal{A}$*,* $a \in P(g.i)$ *and* $t(g.i, a) = g'.i$*; (iii)* $a \in P_E(g.e)$ *and* $t_E(g.e, a) = g'.e$*.*

Having defined the global transition relation, we can proceed to give the concrete semantics describing the behaviour of a system $\mathcal{S}(n)$ composed of n agents and an environment.

Definition 2.12 (Concrete Semantics)**.** *Given a PIIS* $\mathcal{S}$ *and* $n \in \mathbb{N}_1$*, the IIS* $\mathcal{S}(n)$ *is a tuple* $\mathcal{S}(n) = \langle G, g_0, R, V \rangle$*, where* $G = L^n \times L_E$ *is the set of global states,* $g_0 = \langle \iota, \ldots, \iota, \iota_E \rangle$ *is the initial global state,* R *is the global transition relation defined in Definition 2.11 and* $V : G \to 2^{AP \times \mathcal{A}}$ *is the labelling function on the global states defined by* $(p, i) \in V(g)$ *iff* $p \in \mathcal{V}(g.i)$*, for each* $p \in AP$*,* $i \in \mathcal{A}$*, where* AP *is a finite set of atomic propositions.*

Notice that for each atomic proposition p, a copy for each of the n agents is created and a global state is labelled with (p, i) if the agent i is at a local state labelled with p by the template labelling function.

A *path* π is an infinite sequence $\pi = g^0a^0g^1a^1g^2\ldots$ such that $(g^i, a^i, g^{i+1}) \in R$ for every $i \geq 0$. Let $\pi(i)$ denote the i-th state in π. The set of all paths originating from a state g is denoted by $\Pi(g)$.

Specifications for PIISs are typically expressed in the ACTL*K$\backslash X$ logic [Kouvaros and Lomuscio, 2013]. This logic describes the universal fragment of the temporal-epistemic logic CTL*K without the next time operator. Given a set AP of atomic propositions, and a set $\mathcal{A}$ of agents, ACTL*K$\backslash X$ formulae are defined by the following BNF grammar:

$$\phi \quad ::= \quad p \mid \neg p \mid \phi \wedge \phi \mid \phi \vee \phi \mid A(\phi U \phi) \mid A(\phi R \phi) \mid K_i\phi \mid \forall v\colon \phi$$

where $p \in AP$ and $i \in \mathcal{A}$.

Informally, the epistemic modality $K_i\phi$ is read as "agent i knows that ϕ". The temporal modality $A(\phi U \psi)$ stands for "for all paths, at some point ψ holds and before then ϕ is true along the path"; and $A(\phi R \psi)$ denotes "for all paths, ψ holds along the path up to and including the point when ϕ becomes true in the path". We now formally define satisfaction for ACTL*K$\backslash X$ formulas.

Definition 2.13 (Satisfaction of ACTL*K$\backslash X$)**.** *The satisfaction relation $\models$ for an IIS $\mathcal{S}(n)$, and an ACTL*K$\backslash X$ formula ϕ is inductively defined as follows:*

$(\mathcal{S}(n), g) \models A(\phi_1 U \phi_2)$ *iff* *for every $\pi \in \Pi(g)$, for some $i \geq 0$ $(\mathcal{S}(n), \pi(i)) \models \phi_2$ and for all $0 \leq j < i$, $(\mathcal{S}(n), \pi(j)) \models \phi_1$;*

$(\mathcal{S}(n), g) \models A(\phi_1 R \phi_2)$ *iff* *for every $\pi \in \Pi(g)$, for all $i \geq 0$, if $(\mathcal{S}(n), \pi(j)) \not\models \phi_1$, for all $0 \leq j < i$, then $(\mathcal{S}(n), \pi(i)) \models \phi_2$;*

$(\mathcal{S}(n), g) \models K_i\phi$ *iff* *for all $g' \in G$, $g.i = g'.i$ implies $(\mathcal{S}(n), g') \models \phi$;*

We omit the clauses for atomic propositions, conjunction and negation since these are clear.

In [Kouvaros and Lomuscio, 2016], a number of decidability results are given for checking fragments of PIIS against specifications expressed in ACTL*K$\backslash X$. In particular, it is shown that systems constructed using only asynchronous, global-synchronous and role-synchronous

actions always have a cut-off that can be identified (hence, the PMCP is decidable for such systems). It is also shown that sound but incomplete cut-off identification procedures exist for systems constructed using asynchronous and agent-environment actions along with either global-synchronous or multi-role actions. The procedures given work when there is an *agent-environment simulation* in the system which, informally, means that the environment behaves like a mutual exclusion controller with the actions representing access to a shared resource.

The cut-offs considered thus far in the context of modelling swarm systems do not depend on the property in question, so they give a cut-off for *any* formula. This type of cut-off is known as a static cut-off, in contrast to a dynamic cut-off which depends on the property being considered [Kaiser et al., 2010].

Another problem that frequently arises in real-life applications of verification techniques is that of checking a system that has an infinite state space due to the presence of variables such as integers with an infinite domain. Incomplete techniques based on predicate abstraction have been put forward for verifying such systems [Lomuscio and Michaliszyn, 2015]. These have also been combined with results on parameterised systems in order to obtain results on systems with an unbounded number of agents which themselves can be in infinitely many different states [Belardinelli et al., 2017; Kouvaros and Lomuscio, 2017a]. While interesting, this line of work is only tangentially related to the problems we aim to address, so we do not discuss this further.

Another problem of interest is the identification of emergent behaviours [Bonabeau et al., 1999], sometimes referred to as emergent properties, in a parameterised system. Informally, an emergent property is a property that is displayed once a sufficient number of agents is present in the system. For example, a simple local protocol governing the behaviour of a drone in a swarm may result in an overall flocking behaviour, as long as the number of the agents in the formation is larger than a certain threshold [Winfield et al., 2005]. The number of agents needed for a property to be displayed is known as the *emergence threshold* of that

property. Formally:

Definition 2.14 (Emergence)**.** *Given a parameterised system $\mathcal{S}$, a property ϕ is said to be an emergent property of $\mathcal{S}$ if there is some $k \in \mathbb{N}_1$ such that for all $n \geq k$ it is the case that $\mathcal{S}(n) \models \phi$. If this is the case, k is said to be an emergence threshold.*

Emergence identification techniques have been developed for certain classes of systems [Kouvaros and Lomuscio, 2015] but there is still work to be done in this area. The problem of identifying emergent properties is distinct but closely related to the parameterised model checking problem. Indeed, it is possible to translate between the two.

2.3 Fault Tolerance

Inspired by safety-analysis techniques for software verification [Bozzano and Villafiorita, 2007], fault injection methods have been developed for models of swarm systems [Ezekiel and Lomuscio, 2009, 2017]. These rely on adding transitions to the system that represent faulty behaviour (for example, a variable not being updated or being updated erroneously). In addition, extra atomic propositions are added that label where faults have occurred to allow specifications to be written that account for these. These fault injection techniques have been used to model real-life scenarios such as an underwater exploration vehicle that may exhibit faults [Ezekiel et al., 2011].

In [Kouvaros and Lomuscio, 2017b], the work in fault injection was adapted to the PIIS semantics in order to derive results on fault tolerance in a parameterised setting. In particular, the paper addressed the question of checking if it is always the case that a certain property holds in systems of any size even if one in λ of the agents may exhibit faults (where λ is a user-supplied constant). Formally, it addresses the parameterised fault tolerance (PFTP) problem:

Definition 2.15 (PFTP)**.** *Let $\mathcal{S}^f$ be a parameterised system with two agent types: the first*

capturing the possible behaviours of a non-faulty agent and the second capturing the possible behaviours of a faulty agent. Let $\lambda \in \mathbb{N}_1$ *and* ϕ *a specification. Then, the problem is concerned with checking if:*

$$\mathcal{S}((n_n, n_f)) \models \phi \text{ for all } n_n, n_f \in \mathbb{N}_1 \text{ with } n_f = \lfloor n_n/\lambda \rfloor$$

In the paper, an incomplete decision procedure for this problem is presented that works by reducing it to an instance of the PMCP and then using existing incomplete techniques, such as cut-off identification, to solve this. Notice a complete procedure for this problem cannot exist since it would give a decision procedure for the PMCP, which is known to be undecidable [Apt and Kozen, 1986].

The procedure is then used to verify the alpha algorithm with three types of faults injected: direction failures (a robot adopts the wrong movement direction), detection failure (a robot fails to detect some robots in its communication range), and motion failure (a robot fails to move). The specification that is checked is the *connectedness property* [Dixon et al., 2012] that "every non-faulty robot knows it will infinitely often be connected." The algorithm is found to be tolerant to one in four robots being faulty, but not to one in three.

We have since extended this work to address the related issue of efficiently identifying the maximal ratio of agents that can exhibit faulty behaviour whilst still satisfying a specification instead of simply checking the system with a fixed ratio of faulty agents [Kouvaros et al., 2018]. This work is not reported on in this thesis as the systems studied in it are non-probabilistic.

Another line of work addresses fault tolerance in distributed algorithms [John et al., 2013; Aminof et al., 2018]. The semantics used in this context are distinct from those used to model swarm algorithms as we do in this thesis. In particular, restrictions on the communication patterns between processes make them not amenable to the verification of AI systems.

2.4 Summary

In summary, work has been carried out separately in the fields of parameterised and probabilistic model checking. In the former, this has enabled verification of systems with an unbounded number of agents, which has led to the verification of a number of swarm protocols in settings where the number of agents is not known at design time. In the latter, it has enabled verification of meaningful properties of probabilistic systems with a fixed number of agents. While efficient techniques mean that large systems can be verified quickly, they still cannot give any guarantees about systems of sizes larger than those considered.

Limitations. The main gap in existing research, in our opinion, is that the techniques developed give no way to verify that desirable properties of *probabilistic* swarm algorithms hold *regardless of the number of agents in the swarm.* While it is possible to strip away the probabilities in a model of the system and replace them with non-determinism, this will often yield a model in which properties of interest that require probabilities cannot be expressed.

Chapter Three

Unbounded Probabilistic Multi-Agent Systems

In this chapter, we will present two semantics for reasoning about unbounded probabilistic multi-agent systems (UPMAS); that is, stochastic systems that may contain an arbitrarily large number of participants. Our first semantics will be a synchronous one, with agents choosing actions in rounds and then all performing their action simultaneously. In contrast, our second semantics will be an asynchronous one in which agents can act independently (although it will also possible for them to synchronise with other agents through special actions). After presenting these semantics we will go on to define several logics for expressing probabilistic properties of our systems.

The asynchronous semantics we consider second will be more expressive, and enable the modelling of cases where agents typically act independently such as the foraging scenario described in Section 7.2.2. However, the synchronous semantics we describe first will allow us to obtain more decidability results and more naturally model scenarios where interactions occur in fixed rounds, such as the channel jamming scenario considered in Section 7.2.3.

3.1 Synchronous UPMAS

Our model for synchronous probabilistic multi-agent systems (SPMAS) will be made up of a number of agent templates, which capture the possible different behaviours of the participants in the system (there may be arbitrarily many of each type) and an environment, which captures the rest of the state of the system. In particular, our system will be a tuple $\mathcal{S} = \langle \mathcal{T}, E, \mathcal{V}, V_E \rangle$ where $\mathcal{T} = \{T_1, \ldots, T_k\}$ is a finite set of different possible behaviours of participants in the system (noting that for each type of behaviour there may be an unbounded number of agents exhibiting this behaviour), E represents the behaviour of the environment, $\mathcal{V} = \{V_1, \ldots, V_k\}$ defines what atomic propositions hold in different states for each type of agent, and V_E defines what atomic propositions hold for the environment.

We begin by defining the agent templates.

Definition 3.1 (Synchronous Probabilistic Agent Template)**.** *A synchronous probabilistic agent template is a tuple $T_i = \langle S_i, \iota_i, Act_i, P_i, t_i \rangle$ where:*

- *The finite set $S_i \neq \emptyset$ represents the agent's local states.*
- *$\iota_i \in S_i$ is a distinguished initial state.*
- *$Act_i \neq \emptyset$ is a finite set of possible local actions, where we assume a null action $\varepsilon \in Act_i$ exists.*
- *The agent's protocol function $P_i : S_i \to \mathcal{P}(Act_i)$ gives the set of enabled actions in each state. Note we assume that for all $s \in S_i$ we have $\varepsilon \in P_i(s)$, i.e. the null action is always possible.*
- *The agent's transition function $t_i : S_i \times Act_E \times \mathcal{P}(\cup_{j \in k} Act_j) \times Act_i \to Dist(S_i)$ returns a distribution on the agent's next state given its current state, the environment's action, the set of actions performed by all the agents (including the one performed by the agent being considered) and the action performed by this agent at this time-step. We assume*

that, for all s, X and a_E, it is the case that:

$$t_i(s \mid s, a_E, X, \varepsilon) = 1 \tag{3.1}$$

We also assume that, for all s', s, X, a_E and a:

$$t_i(s' \mid s, a_E, X, a) = t_i(s' \mid s, a_E, X \cup \{\varepsilon\}, a) \tag{3.2}$$

Notice that by Equation (3.1) it is the case that agents performing the null action never change state. Further, by Equation (3.2), other agents cannot observe that the null action has been performed. These two conditions ensure that agents can always choose to behave as if they are not there, and not affect the other agents. This will be critical for our procedure to model check strategic properties in Section 6.1.

Notice that we have assumed that there is a unique initial state, rather than a probability distribution on the initial states as is sometimes done in probabilistic model checking literature. This simplifies the presentation of some of our results.

We now define the environment the agents interact with.

Definition 3.2 (Synchronous Probabilistic Environment)**.** *A synchronous probabilistic environment is a tuple $E = \langle S_E, \iota_E, Act_E, P_E, t_E \rangle$ where:*

- *The finite set $S_E \neq \emptyset$ represents the environment's local states.*
- *$\iota_E \in S_E$ is a distinguished initial state.*
- *$Act_E \neq \emptyset$ is a finite set of possible environment actions.*
- *The environment's protocol function $P_E : S_E \to \mathcal{P}(Act_E)$ gives the set of enabled actions in each state.*
- *The environment's transition function $t_E : S_E \times \mathcal{P}(\cup_{j \in k} Act_j) \times Act_E \to Dist(S_E)$ returns a distribution on the environment's next state given its current state, the actions performed by all the agents and the action it performed.*

We assume that, for all s'_E, s_E, X and a_E:

$$t_E(s'_E \mid s_E, X, a_E) = t(s'_E \mid s_E, X \cup \{\varepsilon\}, a_E) \tag{3.3}$$

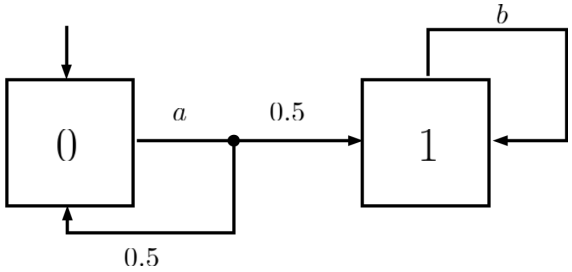
Note that, by Equation (3.3), the environment cannot observe whether a null action was performed by any of the agents, thus agents performing null actions do not affect the transition of the system in any way. Having defined its key components, we now define a probabilistic MAS as consisting of a number of agent templates and an environment, together with labelling functions that define what atomic propositions hold in different states of the system.

Definition 3.3 (Synchronous Probabilistic Multi-Agent System)**.** *A synchronous probabilistic multi-agent system (SPMAS) is a tuple $\mathcal{S} = \langle \mathcal{T}, E, \mathcal{V}, V_E \rangle$, where $\mathcal{T} = \{T_1, \ldots, T_k\}$ is a finite set of probabilistic agent templates, E is an environment, $\mathcal{V} = \{V_1, \ldots, V_k\}$ is a set of agent labelling functions $V_i : S_i \times S_E \to \mathcal{P}(AP)$, and $V_E : S_E \to \mathcal{P}(AP)$ is an environment labelling function.*

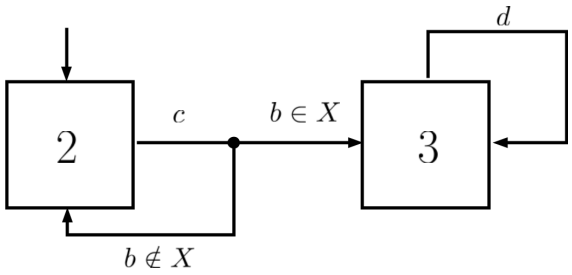
An example SPMAS can be seen in Figure 3.1. Here, agents of the first type begin in an initial state 0 where they can perform an a action, which with equal probability takes them either back to state 0 or to state 1. In state 1, agents can perform the b action that does not change their state. Similarly, agents of the second type begin in state 2. Here, they can perform action c which takes them to state 3 if some other agent has performed b and back to 2 otherwise. In state 3 the agent performs action d and does not change state. The environment is similarly defined except that in order for a transition to change the environment's state the d action must have occurred.

A SPMAS $\mathcal{S}$ gives a description of an infinite number of concrete systems that can be obtained by fixing a number $\bar{n} \in \mathbb{N}_1^k$ of agents in it with $\bar{n}_i$ of type i for each i. We will refer to the j-th agent of type i by (i, j) and denote the set of all agents by

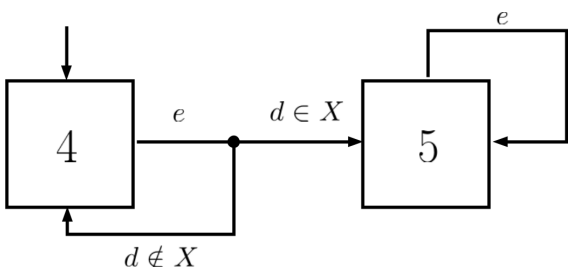
$$\mathcal{A}(\bar{n}) \triangleq \{(i, j) \in \mathbb{N}_1 \times \mathbb{N}_1 \mid 1 \leq i \leq k, 1 \leq j \leq \bar{n}_i\}.$$



(a) An example synchronous probabilistic agent template T_1.



(b) An example synchronous probabilistic agent template T_2.



(c) An example environment E.

Figure 3.1 An example SPMAS. We use X to denote the set of actions performed by all the agents. Note that for clarity the null actions ε are omitted.

For a system of size $\bar{n} \in \mathbb{N}_1^k$, a *global state* $g = \langle \bar{s}_1, \ldots, \bar{s}_k, s_E \rangle$ is a $(k+1)$-tuple. The vector $\bar{s}_i \in (S_i)^{\bar{n}_i}$ gives the local state of the $\bar{n}_i$ agents of type i. The state $s_E \in S_E$ gives the local state of the environment. Thus, the global state encodes all the information to describe the system at a particular instant of time. Let $G_{\bar{n}}$ denote the set of all such global states. For a global state $g = \langle \bar{s}_1, \ldots, \bar{s}_k, s_E \rangle \in G_{\bar{n}}$ we write $g.(i,j)$ to denote the local state of the j-th agent of type i in g, i.e. $(\bar{s}_i)_j$ and $g.E$ to denote the state of the environment in g, i.e. s_E.

An action in this model will correspond to an action of each of the agents, with all agents taking actions at each time step. For a system of size $\bar{n} \in \mathbb{N}_1^k$, a *global action* $a = \langle \bar{a}_1, \ldots, \bar{a}_k, a_E \rangle$ is a $(k+1)$-tuple. The vector $\bar{a}_i \in (Act_i)^{\bar{n}_i}$ gives the actions of the $\bar{n}_i$ agents of type i, while $a_E \in Act_E$ gives the action of the environment. We denote by $Act_{\bar{n}}$ the set of all such global actions. As with global states, we use $a.(i,j)$ to denote the action of the j-th agent of type i in a and $a.E$ to denote the action of the environment in a.

States in the concrete model will be labelled according to the definition given below.

Definition 3.4 (Global Labelling Function)**.** *The global labelling function* $V_{\bar{n}} : G_{\bar{n}} \to \mathcal{P}((AP \times \mathcal{A}(\bar{n})) \cup AP)$ *is defined by:*

$$V_{\bar{n}}(g) \triangleq \{(p,(i,j)) \in AP \times \mathcal{A}(\bar{n}) : p \in V_i(g.(i,j), g.E)\} \cup V_E(g.E)$$

Notice our labelling function creates a new atomic proposition $(p,(i,j))$ for each atomic proposition p and agent (i,j) that holds precisely in the global states where p holds for the local state of agent (i,j). Additionally, atomic propositions p that hold for the environment are added to the set without such a label.

We now define how to obtain a concrete model giving the behaviour of a system in which the number of agents of each type has been fixed. This will be encoded as a Markov decision process (see Section 2.1.3).

Definition 3.5 (Concrete Model)**.** *Given an SPMAS* $\mathcal{S}$*, a concrete model of* $\bar{n}$ *agents for*

$\mathcal{S}$ is an MDP $\mathcal{S}(\bar{n}) = \langle G_{\bar{n}}, \iota_{\bar{n}}, Act_{\bar{n}}, P_{\bar{n}}, V_{\bar{n}} \rangle$, representing the behaviour of a global system composed of $\bar{n}$ agents and the environment, where:

- *The global state $\iota_{\bar{n}} = ((\iota_1, \ldots, \iota_1), \ldots, (\iota_k, \ldots, \iota_k), \iota_E) \in G_{\bar{n}}$ is the initial global state in which every agent and the environment are in their respective initial states.*

- *The protocol function $P_{\bar{n}} : G_{\bar{n}} \to \mathcal{P}(Act_{\bar{n}})$ is defined by:*

$$P_{\bar{n}}(g) \triangleq \{a \in Act_{\bar{n}} \mid \forall (i,j) \in \mathcal{A}(\bar{n}) : a.(i,j) \in P_i(g.(i,j)),\ a.E \in P_E(g.E)\}$$

- *The transition probability $t_{\bar{n}} : G_{\bar{n}} \times Act_{\bar{n}} \to Dist(G_{\bar{n}})$ is given by:*

$$t_{\bar{n}}(g' \mid g, a) \triangleq\ t_E(g'.E \mid g.E, X, a.E) \cdot \prod_{i=1}^{k} \prod_{j=1}^{\bar{n}_i} t_i(g'.(i,j) \mid g.(i,j), a.E, X, a.(i,j))$$

  where $X \triangleq \{a.(i,j) \mid (i,j) \in \mathcal{A}(\bar{n})\}$ is the set of actions performed by at least one agent.

- *The labelling function $V_{\bar{n}}$ is as in Definition 3.4.*

An example of a concrete system can be seen in Figure 3.2. In the initial state, the agent of the first type can perform the a action and with probability 0.5 transition to a local state of 2. The other agent and environment can perform actions c and e, respectively, and do not change state. Once in global state $(1, 2, 4)$ the system behaves deterministically with the second agent transitioning to state 3 after performing action c at the same time as the first agent performs b. Following this, the second agent will perform action d causing the environment to move to state 5.

For a richer example of an SPMAS with more states that models a real-life scenario, we refer the reader to the channel jamming scenario described in Section 7.2.3.

Before proceeding we should check that the definition we have given for the transition probability function is a valid probability distribution. We do so below.

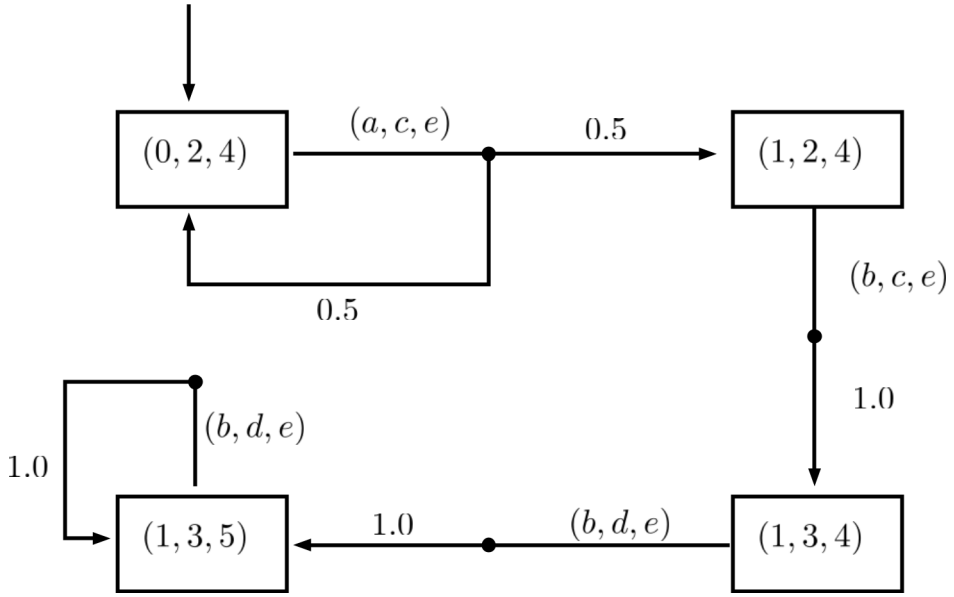


Figure 3.2 The concrete system corresponding to instantiating the SPMAS in Figure 3.1 with one agent of each type. Note that transitions involving the null action ε are omitted for clarity; the full system with null transitions has more choices of transitions and reachable states.

Observation 3.1. *Given an SPMAS $\mathcal{S}$ and $\bar{n} \in \mathbb{N}_1^k$, for all $g \in G_{\bar{n}}$ and $a \in P_{\bar{n}}(g)$ it is the case that:*

$$\sum_{g' \in G_{\bar{n}}} t_{\bar{n}}(g' \mid g, a) = 1$$

Proof. We can compute that

$$\begin{aligned}
&\sum_{g' \in G_{\bar{n}}} t_{\bar{n}}(g' \mid g, a) \\
&\quad = \sum_{g' \in G_{\bar{n}}} \left(t_E(g'.E \mid g.E, X, a.E) \cdot \prod_{i=1}^{k} \prod_{j=1}^{\bar{n}_i} t_i(g'.(i,j) \mid g.(i,j), a.E, X, a.(i,j)) \right) \\
&\quad = \sum_{s_E \in S_E} t_E(s_E \mid g.E, X, a.E) \cdot \sum_{g' \in G_{\bar{n}} : g'.E = s_E} \left(\prod_{i=1}^{k} \prod_{j=1}^{\bar{n}_i} t_i(g'.(i,j) \mid g.(i,j), a.E, X, a.(i,j)) \right) \\
&\quad = \sum_{s_E \in S_E} t_E(s_E \mid g.E, X, a.E) \cdot \prod_{i=1}^{k} \prod_{j=1}^{\bar{n}_i} \left(\sum_{s' \in S_i} t_i(s' \mid g.(i,j), a.E, X, a.(i,j)) \right) \\
&\quad = 1
\end{aligned}$$

with the first equality following by splitting the sum on its change to the state of the environment and the state of the agents and rearranging, and the second equality following by

splitting the sum on the states of the individual agents. Finally, the last equality follows by noting that t_E and all the t_i are valid probability distributions so the sums being multiplied are all 1. □

Notice that before we can reason about the probability of a path occurring in this concrete system we need to resolve the non-determinism of the choice of actions by the agents and environment. We do this with the definition below.

Definition 3.6 (Agent Strategy)**.** *Given a concrete system $\mathcal{S}(\bar{n})$, a strategy for an agent $(i,j) \in \mathcal{A}(\bar{n})$ is a function $\sigma_{i,j} : G_{\bar{n}} \to Dist(Act_i)$ such that for all $g \in G_{\bar{n}}$ it is the case that $\sigma(a \mid g) > 0$ implies $a \in P_i(g.(i,j))$.*

We note that the strategy of agents uses the full global state $G_{\bar{n}}$ and thus our agents have full observability on the state of the concrete system. This differs from the partial observability choice often made in multi-agents systems literature, and is required to obtain our decidability results.

A strategy for the environment σ_E is similarly defined. Given a set of agents, possibly including the environment, $A \subseteq \mathcal{A}(\bar{n}) \cup \{E\}$ we use σ_A to denote a strategy profile giving a strategy for each of these. We use $A^c \triangleq (\mathcal{A}(\bar{n}) \cup \{E\}) \setminus A$ to denote the complement of A.

When we fix a joint strategy σ for all the agents and the environment then the non-determinism in the system is eliminated and the system becomes a discrete time Markov chain as described in Definition 2.7.

3.2 Asynchronous UPMAS

We begin by defining an asynchronous probabilistic multi-agent system. An APMAS is composed of a finite number of *agent templates*, which describe the different possible behaviours of individual agents and an *environment* which captures the behaviour of the other parts of the system.

Definition 3.7 (Asynchronous Probabilistic Agent Template)**.** *An asynchronous probabilistic agent template is a tuple* $T_i = \langle S_i, \iota_i, Act_i, P_i, t_i \rangle$ *where:*

- *The set* S_i *is a finite set of agent local states.*
- $\iota_i \in S_i$ *is a distinguished initial state.*
- $Act_i = A_i \cup AE_i \cup GS$ *is the non-empty set of actions that can be performed by the agents. These may either be asynchronous actions, agent-environment actions or global-synchronous actions. Each type of action implies a different communication pattern between the agents, as will be outlined in Definition 3.11.*
- *The agent's protocol function* $P_i : S_i \to \mathcal{P}(Act_i)$ *defines which actions are enabled at a given state.*
- *The agent's transition function* $t_i : S_i \times Act_i \to Dist(S_i)$ *describes the evolution of the agent's state: given a local state* s *and an action* a *it returns a probability distribution on the state that will be reached when performing action* a *from state* s*.*

The agent template is closely related to MDPs (see Section 2.1.3), suitably extended to encode action types to account for synchronisation purposes.

One particular class of agent templates that is worthwhile to distinguish is those that transition deterministically when performing a global-synchronous action. We formalise this below.

Definition 3.8 (Deterministic-synchronous Agent Template)**.** *A probabilistic agent template* $T_i = \langle S_i, \iota_i, Act_i, P_i, t_i \rangle$ *is said to be deterministic-synchronous if it is the case that whenever* $a \in GS$ *then* $t_i(s' \mid s, a) \in \{0, 1\}$ *for all* $s, s' \in S_i$*.*

Throughout this thesis, we will assume that all the agent templates being considered are deterministic-synchronous. We note that this is not a very severe restriction to our templates since when designing a model for a system one could choose to use asynchronous actions after

a global synchronous action has been performed to model any stochastic behaviour. As we will see in Section 7.2, many realistic scenarios can be modelled even with this restriction.

We now proceed to define the environment that the agents operate in.

Definition 3.9 (Environment)**.** *An environment E is a tuple $E = \langle S_E, \iota_E, Act_E, P_E, t_E \rangle$ where:*

- *S_E is a finite set of local states,*
- *$\iota_E \in S_E$ is a distinguished initial state,*
- *Act_E is a non-empty set of actions $Act_E = A_E \cup \left(\bigcup_{i=1}^{k} AE_i \right) \cup GS$,*
- *The environment protocol P_E is a function $P_E : S_E \to \mathcal{P}(Act_E)$,*
- *$t_E : S_E \times Act_E \to Dist(S_E)$ is a transition function.*

Once again, the environment is similar to an MDP but has been extended to encode different action types. We can now go on to define APMAS, which will be made up of a finite number of agent templates together with an environment and labelling functions which capture what atomic propositions hold at different points of the system's evolution.

Definition 3.10 (APMAS)**.** *An asynchronous probabilistic multi-agent system (APMAS) is a tuple $\mathcal{S} = \langle \mathcal{T}, E, \mathcal{V}, V_E \rangle$, where $\mathcal{T} = \{T_1, \ldots, T_k\}$ is a non-empty and finite set of probabilistic agent templates, E is an environment and $\mathcal{V} = \{V_1, \ldots, V_k\}$ is a set of valuation functions $V_i : S_i \times S_E \to \mathcal{P}(AP)$, one for each agent template, to a set of atomic propositions AP. Finally, $V_E : S_E \to \mathcal{P}(AP)$ is a valuation function for the environment to the atomic propositions.*

Notice that we assume that the sets S_i of local states of each agent template T_i are disjoint, and likewise the sets of actions Act_i are disjoint, with the exception of the global actions GS which are the same for all agents and the environment.

Further, notice that while each agent template has its own valuation function V_i in order to allow us to express local properties of that agent, the valuation functions can all also use the local state of the environment in order to allow us to consider this when deciding what atomic propositions hold. The V_E valuation function allows us to express global properties that are only concerned with the state of the environment.

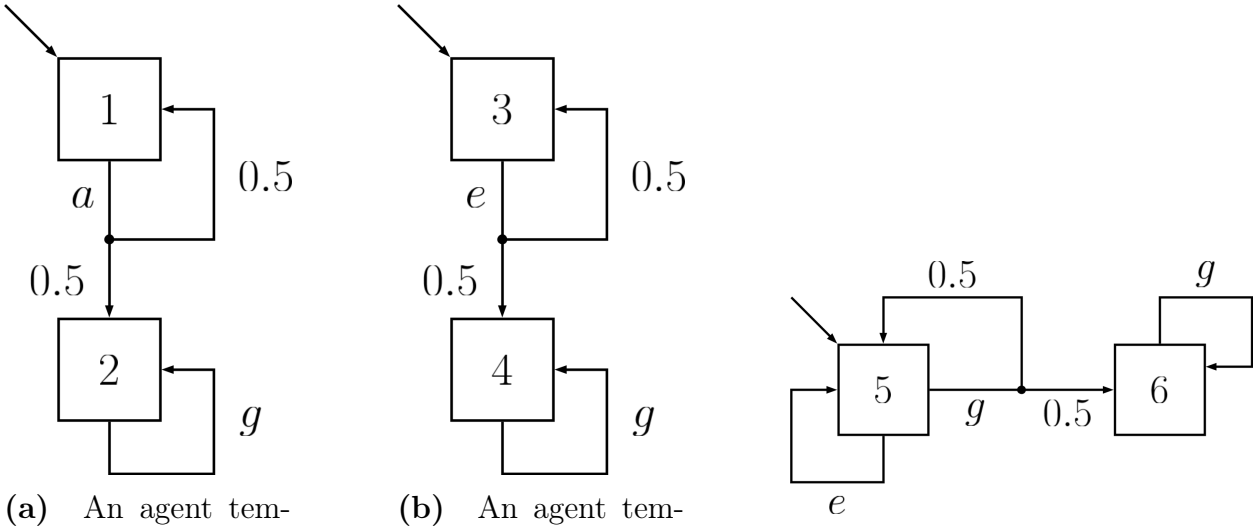
An example APMAS with two agent templates can be seen in Figure 3.3. The agent template T_1 has two states. In the initial state, it can perform the asynchronous a action which with equal probability takes it back to the same state, or takes it to its other state. Once in its second state, the agent can perform the global-synchronous action g. The agent template T_2 is similar except it uses an agent-environment action e instead of an asynchronous one a. Finally, the environment has two states. In its initial state, it can perform the e action or the g one. On performing the g action it can, with probability 0.5, transition to its second state. Once in the second state, it can only perform the g action and remain there.

For a richer example of an APMAS with more states that models a real-life scenario, we refer the reader to the foraging scenario described in Section 7.2.2.

APMASs, as defined above, extend the framework of parameterised interpreted systems (PIISs) presented in [Kouvaros and Lomuscio, 2016] to reason about multi-agent systems composed of an unbounded number of agents. In turn, parameterised interpreted systems extend interleaved interpreted systems (IISs) [Lomuscio et al., 2010]. The framework we introduced above is a modification of PIISs to account for probabilistic behaviour of the agents and the environment.

Each APMAS describes an unbounded collection of concrete systems obtained by choosing a different number of agents in the system. Given an APMAS $\mathcal{S}$ and $\bar{n} \in \mathbb{N}_1^k$, the system $\mathcal{S}(\bar{n})$ of $\bar{n}$ agents is the result of the composition of $\bar{n}_i$ copies of T_i for each $1 \leq i \leq k$ and one environment.

In exactly the same way as for SPMAS (see Section 3.1), we will denote each agent by its identity (i, j) and the set of all agents by $\mathcal{A}(\bar{n})$. As before, we denote by $G_{\bar{n}}$ the set of



(a) An agent template (T_1).

(b) An agent template (T_2).

(c) The environment (E).

Figure 3.3 An example APMAS $\mathcal{S} = \langle\{T_1, T_2\}, E, \{V_1, V_2\}, V_E\rangle$ with two agent templates. The a action is asynchronous, the e action is an agent-environment one, while the g action is global-synchronous.

all global states.

Actions of the global system are of one of four types:

1. *Global-synchronous actions* are performed by all agents and the environment together.

2. *Asynchronous environment actions* are performed by the environment on its own.

3. *Asynchronous agent actions* are performed by one agent on its own, and in the global system are labelled with which agent performed the action.

4. *Agent-environment actions* are performed by one agent together with the environment and in the global system are labelled with which agent performed the action.

We denote by $Act_{\bar{n}}$ the set of all such global actions, i.e.

$$Act_{\bar{n}} \triangleq GS \cup A_E \cup \bigcup_{(i,j)\in\mathcal{A}(\bar{n})} \Big((A_i \cup AE_i) \times \{(i,j)\}\Big)$$

We now go on to define the global protocol, which determines which actions are available at each state in the global system.

Definition 3.11 (Global Protocol)**.** *The global protocol* $P_{\bar{n}} : G_{\bar{n}} \to \mathcal{P}(Act_{\bar{n}})$ *is defined by* $a \in P_{\bar{n}}(g)$ *if and only if*

- (Global-synchronous). *(i)* $a \in GS$*; (ii) for all* $(i,j) \in \mathcal{A}(\bar{n})$*,* $a \in P_i(g.(i,j))$*; (iii)* $a \in P_E(g.E)$*.*
- (Asynchronous environment). *(i)* $a \in A_E$*; (ii)* $a \in P_E(g.E)$*.*
- (Asynchronous agent). *(i)* $a = (a', (i,j)) \in A_i \times \mathcal{A}(\bar{n})$*; (ii)* $a' \in P_i(g.(i,j))$*.*
- (Agent-environment). *(i)* $a = (a', (i,j)) \in AE_i \times \mathcal{A}(\bar{n})$*; (ii)* $a' \in P_i(g.(i,j))$*; (iii)* $a' \in P_E(g.E)$*.*

Thus, global-synchronous actions must be enabled for all agents and the environment. Asynchronous environment actions must be enabled for the environment. Asynchronous agent actions must be enabled for the agents performing them and, finally, agent-environment actions must be enabled for both the agent performing them and the environment.

The next definition gives the global transition function, which defines the probability of transitioning between two states of the global system.

Definition 3.12 (Global Transition Function)**.** *The global transition function* $t_{\bar{n}} : G_{\bar{n}} \times Act_{\bar{n}} \to Dist(G_{\bar{n}})$ *is defined by*

$$t_{\bar{n}}(g' \mid g, a) \triangleq$$

$$\begin{cases} t_E(g'.E \mid g.E, a) \cdot \prod_{i=1}^{k} \prod_{j=1}^{\bar{n}_i} t_i(g'.(i,j) \mid g.(i,j), a) & \text{if } a \in GS \\ t_E(g'.E \mid g.E, a) & \text{if } a \in A_E \text{ and} \\ & \quad \forall (i,j) \in \mathcal{A}(\bar{n}) : g.(i,j) = g'.(i,j) \\ t_i(g'.(i,j) \mid g.(i,j), a') & \text{if } a = (a', (i,j)) \in A \times \mathcal{A}(\bar{n}) \text{ and } g.E = g'.E \text{ and} \\ & \quad \forall (i',j') \in \mathcal{A}(\bar{n}) \setminus \{(i,j)\} : g.(i',j') = g'.(i',j') \\ t_E(g'.E \mid g.E, a') \cdot & \text{if } a = (a', (i,j)) \in AE \times \mathcal{A}(\bar{n}) \text{ and} \\ \quad t_i(g'.(i,j) \mid g.(i,j), a') & \quad \forall (i',j') \in \mathcal{A}(\bar{n}) \setminus \{(i,j)\} : g.(i',j') = g'.(i',j') \\ 0 & \text{otherwise} \end{cases}$$

Thus, the probability of a global transition is always given by multiplying the probability of the agent(s) involved in the action (and, if applicable, the environment) performing their local transition, whilst forcing all those not involved in the action to remain in the same state.

Having given this definition, we should now prove that it does indeed define a valid probability distribution. We do so below.

Observation 3.2. *For any* $g \in G_{\bar{n}}$ *and* $a \in Act_{\bar{n}}$*, it is the case that:*

$$\sum_{g' \in G_{\bar{n}}} t_{\bar{n}}(g' \mid g, a) = 1$$

Proof. We check this for each different type of action a:

- $a \in GS$. Then:

$$\begin{aligned}\sum_{g' \in G_{\bar{n}}} t_{\bar{n}}(g' \mid g, a) &= \sum_{s_E \in S_E} \sum_{g' \in G_{\bar{n}} : g'.E = s_E} \left(t_E(s_E \mid g.E, a) \cdot \prod_{i=1}^{k} \prod_{j=1}^{\bar{n}_i} t_i(g'.(i,j) \mid g.(i,j), a) \right) \\ &= \sum_{s_E \in S_E} t_E(s_E \mid g.E, a) \cdot \sum_{g' \in G_{\bar{n}} : g'.E = s_E} \left(\prod_{i=1}^{k} \prod_{j=1}^{\bar{n}_i} t_i(g'.(i,j) \mid g.(i,j), a) \right) \\ &= \sum_{s_E \in S_E} t_E(s_E \mid g.E, a) \cdot \prod_{i=1}^{k} \prod_{j=1}^{\bar{n}_i} \left(\sum_{s' \in S_i} t_i(s' \mid g.(i,j), a) \right) = 1\end{aligned}$$

  with the first equality following by splitting the sum on its change to the state of the environment and the state of the agents, and the second equality following by splitting the sum on the states of the individual agents.

- $a \in A_E$. Then:

$$\sum_{g' \in G_{\bar{n}}} t_{\bar{n}}(g' \mid g, a) = \sum_{s_E \in S_E} t_E(s_E \mid g.E, a) = 1$$

  with the first equality following from all g' with $g.(i,j) \neq g'.(i,j)$ for some $(i,j) \in \mathcal{A}(\bar{n})$ having $t_{\bar{n}}(g' \mid g, a) = 0$.

- $a = (a', (i,j)) \in A_i \times \mathcal{A}(\bar{n})$. Then:

$$\sum_{g' \in G_{\bar{n}}} t_{\bar{n}}(g' \mid g, a) = \sum_{s \in S_i} t_i(s \mid g.(i,j), a') = 1$$

  with the first equality following again from all other terms of the sum being 0.

- $a = (a', (i,j)) \in AE_i \times \mathcal{A}(\bar{n})$. Then:

$$\begin{aligned}\sum_{g' \in G_{\bar{n}}} t_{\bar{n}}(g' \mid g, a) &= \sum_{s_E \in S_E} \sum_{s \in S_i} t_E(s_E \mid g.E, a') t_i(s \mid g.(i,j), a') \\ &= \sum_{s_E \in S_E} t_E(s_E \mid g.E, a') \left(\sum_{s \in S_i} t_i(s \mid g.(i,j), a') \right) = 1\end{aligned}$$

  with the first equality following by splitting the terms of the sum up between those that change the state of the environment and those that change the state of agent (i,j) and noting that those that change any other component of the state will be 0.

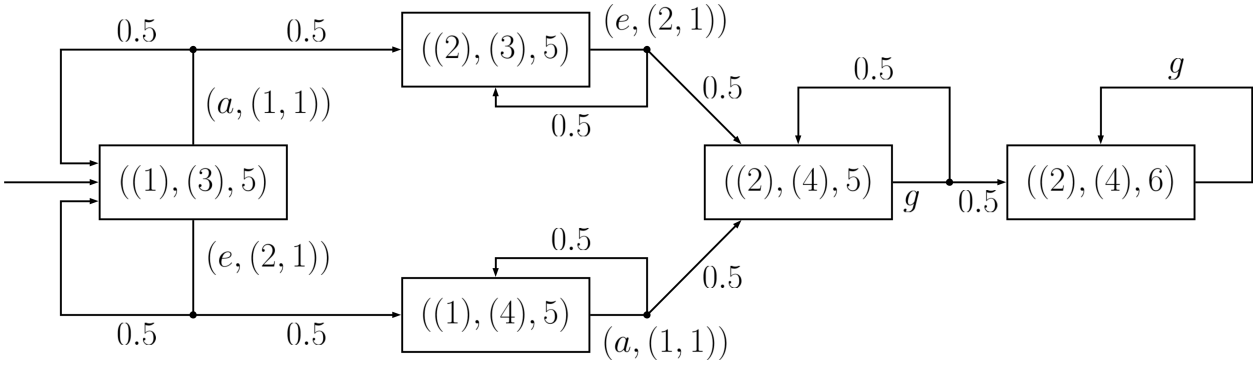


Figure 3.4 An example concrete system $\mathcal{S}((1,1))$ where $\mathcal{S}$ is the APMAS from Figure 3.3.

This covers all cases for the action a, and thus completes our proof. □

Having defined all the necessary components and shown the validity of our definitions, we now go on to combine these to define the concrete system.

Definition 3.13 (Concrete System)**.** *Given an APMAS $\mathcal{S}$ with k agent templates and $\bar{n} \in \mathbb{N}_1^k$, the concrete system of $\bar{n}$ agents is defined by $\mathcal{S}(\bar{n}) = \langle G_{\bar{n}}, \iota_{\bar{n}}, Act_{\bar{n}}, P_{\bar{n}}, t_{\bar{n}}, V_{\bar{n}} \rangle$, where $\iota_{\bar{n}} = \langle (\iota_1, \ldots, \iota_1), \ldots, (\iota_k, \ldots, \iota_k), \iota_E \rangle$ is the initial state where all agents and the environment are in their respective initial states, $P_{\bar{n}}$ is the protocol function from Definition 3.11, $t_{\bar{n}}$ is the transition function from Definition 3.12, and $V_{\bar{n}}$ is the labelling function from Definition 3.4.*

An example of such a concrete system with two agents (one of each type) can be seen in Figure 3.4. In the initial state, either the agent of type 1 can perform the a action or the agent of type 2 can perform the e action. If an agent changes state, then it can no longer perform any action (since the only available action is the g one, but the other agent cannot yet perform it). So, the other agent now performs its action until it changes state, at which point the g action becomes possible. This action may change the state of the environment with probability 0.5, at which point the system reaches state $((2), (4), 6)$ and no further state changes occur (though the g action can continue being performed).

3.3 Specifications for UPMAS

In this section, we will define a few different logics that can be used to express properties of UPMAS. Much of this thesis will focus on giving some decidability results for the parameterised verification problems of these logics against both SPMAS and APMAS.

All of our logics will be constructed to not support nesting of probabilistic operators. This will be crucial for the decidability results presented in Chapters 4 to 6 as the abstract models we present there are focussed on generating abstractions that hold in the initial state, and therefore would not give meaningful results in cases where a probabilistic operator is evaluated on a state other than the starting one.

3.3.1 PLTL

The first logic we consider is *probabilistic linear temporal logic* (PLTL) [Vardi, 1985], which is an extension of LTL [Pnueli, 1977] to handle probabilistic systems and properties. Note we make a slight modification to atomic propositions in order to label them with the agents for which they need to hold, but the logic is otherwise unchanged from its typical presentation [Forejt et al., 2011].

Definition 3.14 (PLTL)**.** *For $a \in AP$ and $i, j \in \mathbb{N}_1$, the probabilistic LTL logic is the set of formulas ϕ defined by the following BNF:*

$$\phi ::= P^{max}_{\bowtie x}[\psi] \;\mid\; P^{min}_{\bowtie x}[\psi] \text{ for } x \in [0,1] \text{ and } \bowtie \in \{\leq, <, \geq, >\}$$

$$\psi ::= \top \;\mid\; (a,(i,j)) \;\mid\; a \;\mid\; \neg\psi \;\mid\; \psi \wedge \psi \;\mid\; X\psi \;\mid\; \psi \; U \; \psi$$

Our formulas have atomic propositions of two types – $(a,(i,j))$ refers to the j-th agent of type i, whilst a refers to the state of the environment. We note that our formulas can only express local properties of the agents or the environment and cannot described global properties such as "half the agents are in a certain state".

We say a formula is $\bar{m}$-indexed, where $\bar{m} \in \mathbb{N}^k$, if $\bar{m}$ is the least tuple such that for any atomic proposition $(a,(i,j))$ that appears in the formula it is the case that $j \leq \bar{m}_i$. Intuitively, this means that satisfaction of the formula is based only on the states of the first $\bar{m}_i$ agents for each type of agent i. Notice that since formulas are finite, every formula has a well-defined index.

The formula $P^{\max/\min}_{\bowtie x}[\psi]$ is read as "with a strategy that tries to maximise/minimise the probability of ψ occurring, this probability is $\bowtie x$". The choice to support both min/max and all operators is not important for later results, but maximises the expressiveness of the logic.

The path formula $X\psi$ is read as "after one time-step ψ holds", whilst $\psi_1 \; U \; \psi_2$ is read as "at some point ψ_2 will hold, and before then ψ_1 must hold."

We will also use standard LTL abbreviations such as $F\psi \equiv \top U \psi$ to mean "at some point ψ holds" and $G\psi \equiv \neg F \neg \psi$ to mean "ψ always holds".

We now formalise the satisfaction relation for this logic.

Definition 3.15 (Satisfaction of PLTL)**.** *Given a concrete system $\mathcal{S}(\bar{n})$ and ϕ an $\bar{m}$-indexed PLTL formula, where $\bar{n}_i \geq \bar{m}_i$ for all i, the satisfaction of ϕ on $\mathcal{S}(\bar{n})$ is inductively defined as follows:*

$$\mathcal{S}(\bar{n}) \models P^{max}_{\bowtie x}[\psi] \quad \textit{iff} \quad \sup_{\sigma \in Strat_{\mathcal{M}}} \mathbf{P}(\{\omega \in IPath_{\mathcal{S}(\bar{n})_\sigma} : \omega \models \psi\}) \bowtie x$$

$$\mathcal{S}(\bar{n}) \models P^{min}_{\bowtie x}[\psi] \quad \textit{iff} \quad \inf_{\sigma \in Strat_{\mathcal{M}}} \mathbf{P}(\{\omega \in IPath_{\mathcal{S}(\bar{n})_\sigma} : \omega \models \psi\}) \bowtie x$$

Satisfaction for path formulae is defined as usual in LTL, i.e.

$$\omega \models \top \qquad \textit{always holds}$$

$$\omega \models (a,(i,j)) \quad \textit{iff} \quad (a,(i,j)) \in V_{\bar{n}}(\omega_0)$$

$$\omega \models a \quad \textit{iff} \quad a \in V_{\bar{n}}(\omega_0)$$

$$\begin{aligned}
&\omega \models \neg\psi && \textit{iff} \quad \omega \not\models \psi \\
&\omega \models \psi_1 \wedge \psi_2 && \textit{iff} \quad \omega \models \psi_1 \textit{ and } \omega \models \psi_2 \\
&\omega \models X\psi && \textit{iff} \quad \omega(1) \models \psi \\
&\omega \models \psi_1 U \psi_2 && \textit{iff} \quad \textit{for some } i \geq 0,\ \omega(i) \models \psi_2 \textit{ and for all } 0 \leq j < i,\ \omega(j) \models \psi_1
\end{aligned}$$

Notice that the set of paths defined by the path formula ψ is always measurable (see Corollary 2.4 in [Vardi, 1985]), so the probability in our definition is well-defined.

For an example PLTL formula, consider a collective transport scenario [Ferrante et al., 2013] where a group of robots (all of the same type) aims to collaborate to move an object to a destination. In this scenario we might want to consider a property such as:

$$P^{min}_{>0.7}[F\text{objectAtDestination}]$$

which says that even with a strategy that tries to minimise the chances of this happening, there is still a probability of over 0.7 that the object will eventually reach its destination. The index of this formula is (0) since it refers to no agents.

For another example, consider an opinion formation scenario [de Oca et al., 2011] where a group of agents (of two different types) must agree on a choice from a set of actions. In this scenario it might be desirable to consider a property such as:

$$P^{max}_{\leq 0.1}[G(\neg(\text{decisionReached}, (2,1))]$$

which says that even with a strategy that tries to maximise the chances of this happening, the probability of agent $(2,1)$ never reaching a decision is at most 0.1. The index of this formula is $(0,1)$ since it refers to no agents of the first type, and the first agent of the second type.

3.3.2 PLTL$_\mathbf{k}$

Having defined the PLTL logic, we also define a similar logic, which we will call PLTL$_\text{k}$. This will be a variation of PLTL to only contain operators that look a bounded distance into

traces. This restriction will be important as the abstract models we introduce later in this thesis will be able to almost exactly simulate the behaviour of arbitrarily large systems only for a finite number of steps. Thus, we will be able to develop a complete decision for PLTL_k but not for PLTL.

Note that while our logic is related to the logic PLTL_f [Maggi et al., 2019], the two are distinct in that our logic uses operators that explicitly only consider a finite part of the trace, while the logic presented there interprets the standard PLTL operators on finite traces.

Definition 3.16 (PLTL_k)**.** *For $a \in AP$ and $i, j, k \in \mathbb{N}_1$, the $PLTL_k$ logic is the set of formulas ϕ defined by the following BNF:*

$$\phi ::= P_{\leq}^{max}[\psi] \text{ for } x \in (0,1)$$
$$\psi ::= \top \mid (a,(i,j)) \mid a \mid \neg\psi \mid \psi \wedge \psi \mid X\psi \mid \psi \; U^{<k} \; \psi$$

The PLTL_k logic is similar to PLTL, except that the unbounded until operator U has been replaced by a bounded-time version $U^{<k}$ (this could also be a non-strict $U^{\leq k}$ as is sometimes found in other probabilistic model checking literature without affecting any of the results presented later in this thesis).

Further, instead of supporting more general formulas of the form $P_{\bowtie}^{\max/\min}[\psi]$, we only support formulas of the form $P_{\leq}^{\max}[\psi]$. Here, the choice to support only the weak $\leq$ is crucial for the decidability result presented in Chapter 4, as our abstract models can only approximate concrete systems and are therefore not suited to verifying properties with strict inequalities.

The path formula $\psi_1 \; U^{<k} \; \psi_2$ is read as "at some point within k time steps the formula ψ_2 holds, and until then the formula ψ_1 holds". The other cases are interpreted as in PLTL. This new logic is less expressive than PLTL since it cannot describe properties of infinite paths such as requiring an event to occur at some point arbitrarily far in the future. The index $\bar{m}$ of a PLTL_k formula is defined as for PLTL. We now define satisfaction for this logic.

Definition 3.17 (Satisfaction of $\mathrm{PLTL_k}$)**.** *Given a concrete system $\mathcal{S}(\bar{n})$ and ϕ an $\bar{m}$-indexed $PLTL_k$ formula, where $\bar{n}_i \geq \bar{m}_i$ for all i, the satisfaction of ϕ on $\mathcal{S}(\bar{n})$ is inductively defined as in Definition 3.15 with the case for the new $U^{<k}$ operator being given by*

$$\omega \models \psi_1 U^{<k} \psi_2 \quad \textit{iff} \quad \textit{for some } 0 \leq i < k,\ \omega(i) \models \psi_2 \textit{ and for all } 0 \leq j < i,\ \omega(j) \models \psi_1$$

We now define precisely how far in the future a $\mathrm{PLTL_k}$ formula describes. We will denote this the time bound of the formula.

Definition 3.18 (Time Bound)**.** *The time bound of a $PLTL_k$ formula is inductively defined by:*

$$\begin{aligned}
tb(P^{max}_{\leq x}[\psi]) &\triangleq tb(\psi) \\
tb(\top) &\triangleq 0 \\
tb((a,(i,j))) &\triangleq 0 \\
tb(a) &\triangleq 0 \\
tb(\neg\psi) &\triangleq tb(\psi) \\
tb(\psi_1 \wedge \psi_2) &\triangleq max(tb(\psi_1), tb(\psi_2)) \\
tb(X\psi) &\triangleq tb(\psi) + 1 \\
tb(\psi_1 U^{<k} \psi_2) &\triangleq max(tb(\psi_1), tb(\psi_2)) + k
\end{aligned}$$

For an example of a $\mathrm{PLTL_k}$ formula, consider an aggregation protocol [Nembrini, 2005] in which a group of agents (all of the same type) aim to move around so they are all within communication range of each other ("connected").

A property of interest in such a scenario is that, irrespective of the choice of strategy, a certain agent is connected to the swarm within 10 time steps with a probability of over 0.9. This can be expressed by a $\mathrm{PLTL_k}$ formula such as

$$P^{\min}_{>0.9}[F^{\leq 10}(connected, (1,1))],$$

where *connected* is an atomic proposition holding when an agent is connected. This formula has index (1) since it refers to only the first agent. It has time bound 10 since in order to evaluate it the first 10 states of a trace would need to be considered.

3.3.3 P[ATL*]

Finally, we will consider specifications that enable the verification of strategic properties of the system. To do this, we introduce a fragment of PATL* [Chen and Lu, 2007], which we call P[ATL*].

Definition 3.19 (P[ATL*]). *For $a \in AP$ and $i, j \in \mathbb{N}_1$, the P[ATL*] logic is the set of formulas ϕ defined by the following BNF:*

$$\phi ::= \langle\langle A \rangle\rangle P_{\bowtie x}[\psi] \text{ for } x \in [0,1] \text{ and } \bowtie \in \{\leq, <, \geq, >\}$$
$$\psi ::= \top \mid (a,(i,j)) \mid a \mid \neg\psi \mid \psi \wedge \psi \mid X\psi \mid \psi U \psi,$$

where $A \subset (\mathbb{N}_1 \times \mathbb{N}_1) \cup \{E\}$ is a finite set of agents (and possibly the environment).

Notice this corresponds to the fragment of PATL* in which we only allow one strategy operator at the top of the formula. While not as general as full PATL*, this fragment is still expressive enough to allow us to verify properties of interest in multi-agent systems, as we will see in Chapter 7. The index $\bar{m}$ of a P[ATL*] formula is defined as for PLTL.

The formula $\langle\langle A \rangle\rangle P_{\bowtie r}[\psi]$ is read as "agents A have a strategy to ensure that ψ occurs with probability $\bowtie r$". The path formulas ψ are interpreted as in PLTL.

Consider again the opinion formation scenario from earlier [de Oca et al., 2011] where a group of agents (of two different types) must agree on a choice from a set of actions. Then, the P[ATL*] formula

$$\langle\langle (1,1), E \rangle\rangle P_{\geq 0.5}[G\neg(\mathit{decisionReached}, (2,1))]$$

represents that agent (1,1) and the environment have a strategy that ensures with probability at least 0.5 that agent (2,1) does not reach a decision. The index of this formula is $(1,1)$ since it refers to the first agent of each of the two types.

We now formally define the satisfaction relation for this logic.

Definition 3.20 (Satisfaction)**.** *Given a concrete system $\mathcal{S}(\bar{n})$ and an $\bar{m}$-indexed P[ATL*] formula $\phi = \langle\langle A\rangle\rangle P_{\bowtie x}[\psi]$ with $\bar{m}_i \leq \bar{n}_i$ for all i, we say the formula is satisfied in $\mathcal{S}(\bar{n})$, denoted by $\phi \models \mathcal{S}(\bar{n})$ iff there is some strategy profile σ_A for the agents in A such that for all strategy profiles σ_{A^c} for the agents in A^c it is the case that:*

$$\mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\{\omega \in IPath_{\mathcal{S}(\bar{n})_\sigma}(\iota) : \omega \models \psi\}) \bowtie x$$

where $\mathcal{S}(\bar{n})_\sigma$ denotes the DTMC obtained by fixing the joint strategy given by σ_A and σ_{A^c} in $\mathcal{S}(\bar{n})$. Satisfaction of path formulas is defined as in PLTL (see Definition 3.15).

Note that the strategy profile associates with each agent a strategy in the sense of Definition 3.6, i.e. with full observability of the global state.

3.4 Summary

This chapter introduced SPMAS and APMAS, two distinct semantics for modelling unbounded probabilistic multi-agent systems (UPMAS) as well as three specification logics for describing properties of these systems: PLTL, PLTL$_\mathrm{k}$, and P[ATL*].

The first semantics, *synchronous probabilistic multi-agent systems* (SPMAS) involves agents interacting in distinct rounds, with agents simultaneously choosing an action and then the result of the actions being resolved for all agents. Given an SPMAS $\mathcal{S}$ and a vector $\bar{n}$ fixing the number of agents of each type, we defined how to obtain an MDP $\mathcal{S}(\bar{n})$ that describes the behaviour of the system with precisely $\bar{n}_i$ agents of each type i.

The second semantics introduced in this chapter, *asynchronous probabilistic multi-agent systems* (APMAS) involves different types of actions: asynchronous environment actions, asynchronous agent actions, agent-environment actions, and global-synchronous actions. Depending on the type of action, different combinations of agents and/or the environment participate in the resulting transition. Once again, we defined the MDP $\mathcal{S}(\bar{n})$ describing the behaviour of concrete systems.

We then introduced three specification logics for reasoning about properties of concrete systems. Our first logic, PLTL, enables the verification of temporal properties on traces of possibly infinite length. We then restricted this logic to a less expressive fragment, which we called PLTL_{k}. This logic only contains operators that look a bounded distance into traces. Finally, we defined a logic $\text{P}[\text{ATL}^*]$ which enables the verification of strategic properties, i.e. specifications describing what temporal properties a coalition of agents can enforce.

Chapter Four

Verifying Bounded-Time Properties

In this chapter, we will define the parameterised model checking problem (PMCP) for bounded-time properties. We will then present an abstract model that gives a finite representation of all possible behaviours of an SPMAS of any size. Using this abstract model, we will define a decision procedure for our PMCP, and prove that this procedure is sound and complete.

The material presented in this chapter has been heavily adapted to fit the rest of this thesis, but is loosely based on research first presented in our 2018 IJCAI paper [Lomuscio and Pirovano, 2018].

4.1 Parameterised Model Checking Problem

In this section we define the parametrised model checking problem that we will consider for the rest of the chapter. In particular, this will be the problem of checking whether a PLTL_{k} formula holds in an SPMAS of any size. We define this below.

Definition 4.1 (PLTL_{k} on SPMAS)**.** *Given an SPMAS $\mathcal{S}$ and an $\bar{m}$-indexed $PLTL_k$ formula ϕ, the parameterised model checking problem (PMCP) involves establishing whether it is the case that $\mathcal{S}(\bar{n}) \models \phi$ for all $\bar{n}$ with $\bar{n}_i > \bar{m}_i$ for all i. We write $\mathcal{S} \models \phi$ if this is the case.*

The rest of this chapter will be devoted to building a sound and complete procedure for solving the above problem, which will work by constructing an abstract model that captures all possible behaviours of the system in instantiations of any size.

4.2 Abstract Model

In this section, we develop our abstract model whose states have two components: the first captures the state of the first $\bar{m}$ agents which are referred to in the formula, the second records the set of states that arbitrarily many other agents are in. Note that the number of agents in each state is not recorded for these; we abstract away the exact number and instead just track whether there are some agents in a certain state or none. This abstract model is inspired by the counter abstraction models defined in [Pnueli et al., 2002]. However, these are adapted to the different semantics developed here.

Definition 4.2 (Abstract Model of an SPMAS)**.** *Given an SPMAS $\mathcal{S}$, the abstract model of $\bar{m}$ agents for the system $\mathcal{S}$ is an MDP $\hat{\mathcal{S}}(\bar{m}) = \langle \hat{S}_{\bar{m}}, \hat{\iota}_{\bar{m}}, \hat{Act}_{\bar{m}}, \hat{P}_{\bar{m}}, \hat{V}_{\bar{m}} \rangle$, where:*

- *The set of possible global states $\hat{S}_{\bar{m}} \triangleq S_{\bar{m}} \times \mathcal{P}(\cup_{i=1}^{k} S_i)$ has two components. The first records the state of the first $\bar{m}$ agents and the environment, the second is a set recording all local states of all other agents.*
- *The initial state is $\hat{\iota} \triangleq (\iota_{\bar{m}}, \{\iota_1, \ldots, \iota_k\})$.*
- *The set of possible actions is $\hat{Act}_{\bar{m}} \triangleq Act_{\bar{m}} \times \mathcal{P}(\cup_{i=1}^{k} Act_i)$.*
- *The protocol function $\hat{P}_{\bar{m}} : \hat{S}_{\bar{m}} \to \mathcal{P}(\hat{Act}_{\bar{m}})$ is defined by:*

$$\bar{P}_m(g, X) \triangleq \{(a, Y) \in \hat{Act}_{\bar{m}} \mid a \in P_{\bar{m}}(g), \forall a' \in Y \exists i \in \dot{k}, s \in X : a' \in P_i(s)\}$$

- *The transition probability* $\hat{t}_{\bar{m}} : \hat{S}_{\bar{m}} \times \hat{Act}_{\bar{m}} \to Dist(\hat{S}_{\bar{m}})$ *is given by:*

$$\hat{t}_{\bar{m}}((g', X') \,|(g, X), (a, Y)) \triangleq$$
$$t_E(g'.E \,|g.E, Z, a.E) \times \prod_{i=1}^{k} \prod_{j=1}^{\bar{m}_i} t_i(g'.(i,j) \mid g.(i,j), a.E, Z, a.(i,j))$$
$$\times \begin{cases} 1 \text{ if } X' = \mathcal{R}_{X,(a,Y)} \\ 0 \text{ otherwise} \end{cases}$$

  where $Z \triangleq \{a.(i,j) \;\mid\; (i,j) \in \mathcal{A}(\bar{m})\} \cup Y$ *is the set of actions performed by at least one agent in either the first component or the second and*

$$\mathcal{R}_{X,(a,Y)} = \bigcup_{j=1}^{k} \{s' \in S_j \mid \exists s \in X \exists a \in \hat{Y} \cap P(s) : t_i(s' \mid s, a.E, Z, a) > 0\}$$

  is the set of states that could be reached in one step by performing the actions Y *from the set* X*.*

- *The labelling function* $\hat{V}_{\bar{m}} : \hat{S}_{\bar{m}} \to \mathcal{P}(AP \times \mathcal{A}(\bar{m}))$ *is given by:*

$$\hat{V}_{\bar{m}}(g, X) \triangleq V_{\bar{m}}(g)$$

  The function $\hat{V}_{\bar{m}}$ *discards information not pertaining to the first* $\bar{m}$ *agents, since this is not needed to evaluate an* $\bar{m}$*-indexed formula.*

Intuitively, the second component captures the states that it is possible for one of the agents after the first $\bar{m}$ to be in. Note that since we want to capture in the abstract model the behaviour of arbitrarily large systems, we assume in our model that there are many agents in each state and, therefore, any state that has a non-zero probability of being reached by an agent will be reached by at least one of them with probability 1.

Given an abstract state $\hat{g} = (g, X) \in S_{\bar{m}}$, we write $\hat{g}.(i,j) \triangleq g.(i,j)$ for the local state of the j-th agent of type i as before. We also write $\hat{g}.abs \triangleq X$ for the state of the abstract agents in the second component.

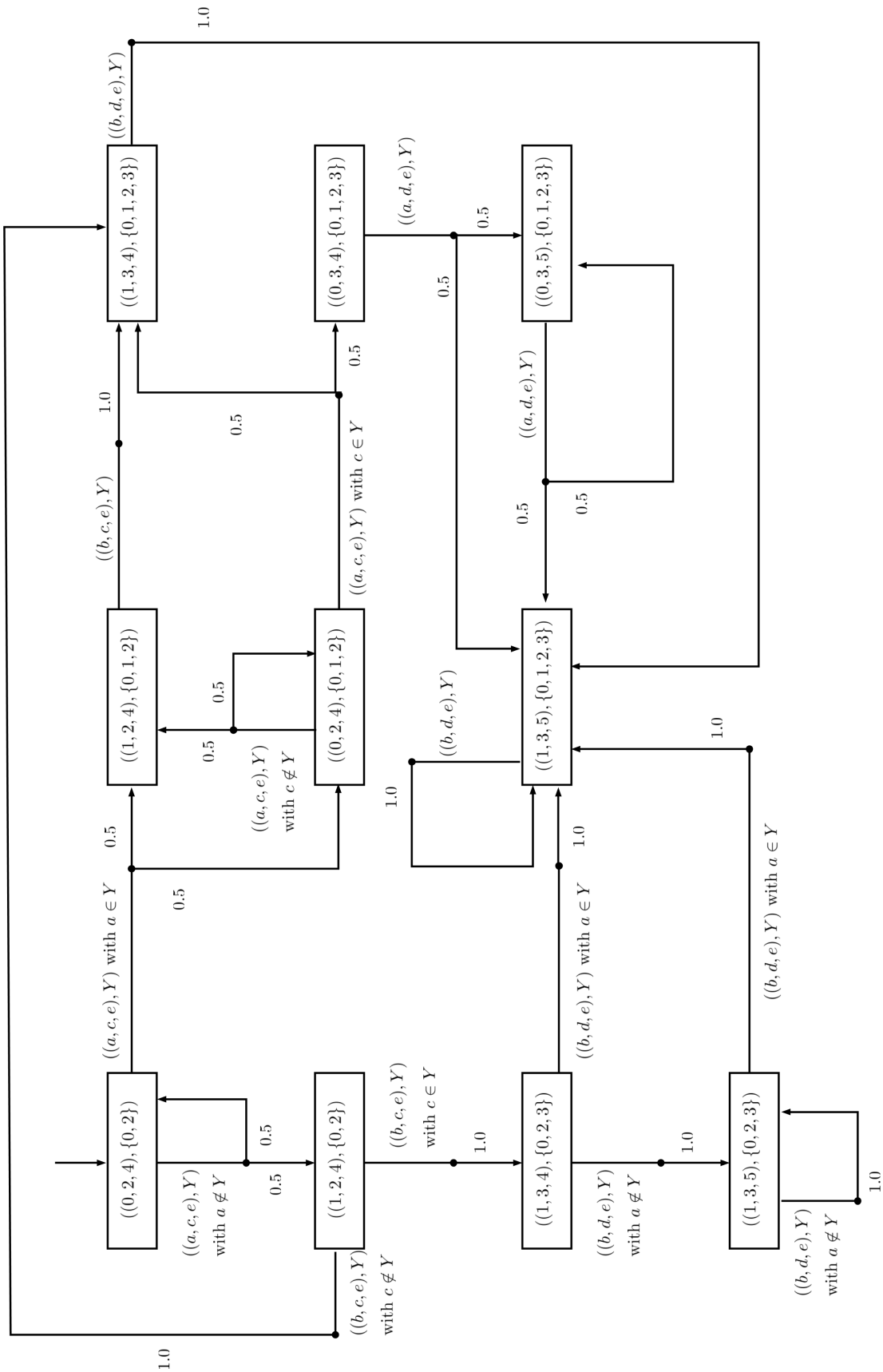


Figure 4.1 The abstract model with one agent of each type for the SPMAS in Figure 3.1. We omit the null actions ε for clarity.

An example abstract model can be seen in Figure 4.1. In the initial state, the first agent can perform action a and with probability 0.5 may change its state to 1. The second agent can only perform action c and does not change state. Similarly, the environment can only perform action e and remain in its state. The remaining agents could choose to either perform a or not. If they perform a then the abstract model assumes that the state 1 is reached by at least one of them, and the second component is updated to $\{0, 1, 2\}$. Otherwise, the second component remains unchanged. The rest of the states in the abstract model have similarly defined transitions.

Notice that when defining a joint strategy for the abstract model, this also includes a choice of the set of actions performed by the abstract agents in the second component of the state.

Before proceeding, we should check that the transition function for our abstract model is always a valid probability distribution. We do so now.

Observation 4.1. *For any $\hat{s} \in \hat{S}_{\bar{m}}$ and $\hat{a} \in \hat{Act}_{\bar{m}}$, it is the case that:*

$$\sum_{\hat{s}' \in \hat{S}_{\bar{m}}} \hat{t}_{\bar{m}}(\hat{s}' \mid \hat{s}, \hat{a}) = 1$$

Proof. Let $\hat{s} = (s, X) \in \hat{S}_{\bar{m}}$ and $\hat{a} = (a, Y) \in \hat{Act}_{\bar{m}}$. Then:

$$\begin{aligned}
\sum_{\hat{s}' \in \hat{S}_{\bar{m}}} \hat{t}_{\bar{m}}(\hat{s}' \mid \hat{s}, \hat{a}) &= \sum_{s' \in S_{\bar{m}}} \left(t_E(s'.E \mid s.E, Z, a.E) \times \prod_{i=1}^{k} \prod_{j=1}^{\bar{m}_i} t_i(s'.(i,j) \mid s.(i,j), a.E, Z, a.(i,j)) \right) \\
&= \sum_{s_E \in S_E} t_E(s_E \mid g.E, Z, a.E) \times \sum_{g' \in G_{\bar{n}} : g'.E = s_E} \left(\prod_{i=1}^{k} \prod_{j=1}^{\bar{n}_i} t_i(g'.(i,j) \mid g.(i,j), a.E, Z, a.(i,j)) \right) \\
&= \sum_{s_E \in S_E} t_E(s_E \mid g.E, Z, a.E) \times \prod_{i=1}^{k} \prod_{j=1}^{\bar{n}_i} \left(\sum_{s' \in S_i} t_i(s' \mid g.(i,j), a.E, Z, a.(i,j)) \right) \\
&= 1
\end{aligned}$$

where $Z \triangleq \{a.(i,j) \mid (i,j) \in \mathcal{A}(\bar{m})\} \cup Y$. The first equality follows from the definition of $\hat{t}_{\bar{m}}$ in Definition 4.2, removing the terms which are 0. The rest follows similarly to Observa-

tion 3.1, except that instead of just the actions of the concrete agents being used in the set Z, the actions of the abstract ones are used too. □

In the next subsection, we will show that our abstract system is capable of simulating larger systems and give a result showing what this implies for the parametric model checking of PLTL_k formulae. Following this, in the second subsection we will show a converse result – the behaviour of large systems approaches that of the abstract system. Again, this will give us a result that enables the parametrised model checking of PLTL_k formulae.

4.2.1 Simulating Larger Systems

In order to achieve our goal for this subsection, we need to formalise in what sense our abstract model captures the same paths as larger concrete systems. As a stepping stone to doing so, we first define how a concrete state can be mapped to an abstract one.

Definition 4.3 (State Abstraction)**.** *Let $\bar{n}, \bar{m} \in \mathbb{N}_1^k$ with $\bar{n}_i > \bar{m}_i$ for all i. Then, the abstraction map on states $\lambda_{\bar{n},\bar{m}} : S_{\bar{n}} \to \hat{S}_{\bar{m}}$ is given by:*

$$\lambda_{\bar{n},\bar{m}}(\bar{s}_1, \ldots, \bar{s}_k, s_E) \triangleq ((\bar{s}'_1, \ldots, \bar{s}'_k, s_E), X) \text{ where}$$

$$\bar{s}'_i \triangleq (\bar{s}_{i,1}, \ldots, \bar{s}_{i,\bar{m}_i}) \text{ and } X \triangleq \bigcup_{i=1}^{k} \{\bar{s}_{i,\bar{m}_i+1}, \ldots, \bar{s}_{i,\bar{n}_i}\}$$

Intuitively, for each type of agent i this map preserves the exact state of the first $\bar{m}_i$ such agents in the first component. The states of the remaining agents are projected into the set in the second component, thus preserving which states are present in one or more agents but discarding the information on precisely how many agents are in each state. We now go on to define a similar abstraction notion for actions.

Definition 4.4 (Action Abstraction)**.** *Let $\bar{n}, \bar{m} \in \mathbb{N}_1^k$ with $\bar{n}_i > \bar{m}_i$ for all i. Then, we define*

the action abstraction map $\lambda_{\bar{n},\bar{m}} : Act_{\bar{n}} \to \hat{Act}_{\bar{m}}$ *by:*

$$\lambda_{\bar{n},\bar{m}}(a) \triangleq ((\bar{a}_1, \ldots, \bar{a}_k), X)$$
$$\text{where } \bar{a}_i = (a.(i,1), \ldots, a.(i,\bar{m}_i))$$
$$\text{and } X = \bigcup_{i=1}^{k} \{a.(i,\bar{m}_i+1), \ldots, a.(i,\bar{n}_i)\}$$

Once again, for each type of agent i this map preserves the exact action of the first $\bar{m}_i$ such agents while projecting the rest into a set, discarding the information about precisely how many agents performed each action.

Another notion that will be useful for us is defining when a concrete state agrees with the first component of an abstract one. This will be important since the first $\bar{m}$ agents in the first component are the ones that are considered when evaluating a formula. We define this notion below.

Definition 4.5 (Concrete Equivalence)**.** *Let* $\bar{n}, \bar{m} \in \mathbb{N}_1^k$ *with* $\bar{n}_i > \bar{m}_i$ *for all* i*. Then, we say an abstract state* $\hat{g} \in \hat{S}_{\bar{m}}$ *and a concrete state* $g \in S_{\bar{n}}$ *agree on the concrete states if:*

$$\hat{g}.E = g.E \text{ and } \hat{g}.(i,j) = g.(i,j) \text{ for all } 1 \leq i \leq k \text{ and } 1 \leq j \leq m_i$$

When this is the case, we write $\hat{g} =_{conc} g$*.*

Note that this requires the first $\bar{m}_i$ agents for each type i to have the same state, but says nothing about the state of the remaining agents. This brings us to our first technical lemma, which shows that the probability of a transition in the abstract model is precisely the sum of the concrete transitions it could represent.

Lemma 4.1. *Let* $\bar{n}, \bar{m} \in \mathbb{N}_1^k$ *with* $\bar{n}_i > \bar{m}_i$ *for all* i*. Let* $g \in S_{\bar{n}}$ *and* $a \in P_{\bar{n}}(g)$*. Then, letting* $\hat{g} = \lambda_{\bar{n},\bar{m}}(g)$ *and* $\hat{a} = \lambda_{\bar{n},\bar{m}}(a)$*:*

$$\hat{t}_{\bar{m}}(\hat{g}' \mid \hat{g}, \hat{a}) = \sum_{g' \in S_{\bar{n}} : g' =_{conc} \hat{g}'} t_{\bar{n}}(g' \mid g, a)$$

whenever $\hat{g}'.abs = \mathcal{R}_{\hat{g}.abs,\hat{a}}$*.*

Proof. Starting from the right-hand side, we get:

$$\begin{aligned}\sum_{g' \in S_{\bar{n}} : g' =_{conc} \hat{g}'} t_{\bar{n}}(g' \mid g, a) &= t_E(\hat{g}'.E \mid \hat{g}.E, X, a.E) \times \prod_{i=1}^{k} \prod_{j=1}^{\bar{m}_i} t_i(\hat{g}'.(i,j) \mid \hat{g}.(i,j), a.E, X, a.(i,j)) \\ &\quad \times \sum_{g' \in S_{\bar{n}} : g' =_{conc} \hat{g}'} \left(\prod_{i=1}^{k} \prod_{j=\bar{m}_i}^{\bar{n}_i} t_i(g'.(i,j) \mid g.(i,j), a.E, X, a.(i,j)) \right) \\ &= t_E(\hat{g}'.E \mid \hat{g}.E, X, a.E) \times \prod_{i=1}^{k} \prod_{j=1}^{\bar{m}_i} t_i(\hat{g}'.(i,j) \mid \hat{g}.(i,j), a.E, X, a.(i,j)) \\ &= \hat{t}_{\bar{m}}(\hat{g}' \mid \hat{g}, \hat{a})\end{aligned}$$

with the first equality following by applying the definitions and re-arranging, the second equality following by noting that all the t_i's are valid transition functions and therefore sum to 1, and the final equality following by applying the definitions again. □

While useful, this lemma alone does not yet allow us to reason about the satisfaction of formulae. To do this, we need a result establishing a link between the abstract model and the concrete ones for not just individual transitions but entire paths. Before we can do this, we need a few more technical definitions. Our first definition, below, establishes precisely which concrete paths are represented by paths in the abstract model.

Definition 4.6 (Path Simulation)**.** *Let $\bar{n}, \bar{m} \in \mathbb{N}_1^k$ with $\bar{n}_i > \bar{m}_i$ for all i. Then, we say an abstract path $\hat{\rho} = \hat{s}_1 \hat{a}_1 \hat{s}_2 \ldots \in IPath_{\hat{\mathcal{S}}(\bar{m})}$ simulates a concrete path $\rho = s_1 a_1 s_2 \ldots \in IPath_{\mathcal{S}(\bar{n})}$ if for all $i \in \mathbb{N}_1$:*

(i) $\hat{s}_i =_{conc} s_i$

(ii) $\hat{a}_i = \lambda_{\bar{m},\bar{n}}(a_i)$

(iii) $\hat{s}_{i+1}.abs = \mathcal{R}_{\hat{s}_i.abs, \hat{a}_i}$

If this is the case, we write $\hat{\rho} \sim \rho$.

We observe that if $\hat{\rho} \sim \rho$ then ρ and $\hat{\rho}$ satisfy the same LTL formulas provided these are at most $\bar{m}$-indexed. This follows trivially from condition (i) since the formula only references atomic propositions that appear in the concrete state.

Note that the above definition is a relation rather than a function since each path in the abstract model can represent many paths in the concrete model. We restrict the definition above to finite paths in the obvious way (i.e. requiring the same conditions to hold up to the length of the path).

Recall that our main goal in this subsection is to show an abstract system of size $\bar{m}$ can simulate the behaviour of larger systems. In order to achieve this, we need to define a way in which given a strategy for a system of size $\bar{n}$ (with $\bar{n}_i > \bar{m}_i$ for all i), we can simulate this in the abstract system of size $\bar{m}$.

Before we go on to give this definition, we note that each path in the abstract system can represent many paths in the concrete system. Thus, when choosing how to act based on a path in the abstract system, the equivalent strategy cannot simply lookup the choice that our original strategy would've made in the concrete system. Instead, it has to look up all the possible choices for the different concrete paths that our abstract path could represent and build a probability distribution according to these. The precise definition of this follows. As we will see Lemma 4.2 this definition achieves the desired outcome of having paths in the abstract system being as likely as the sum of the concrete paths that they represent.

Definition 4.7 (Equivalent Strategy)**.** *Let $\bar{n}, \bar{m} \in \mathbb{N}_1^k$ with $\bar{n}_i > \bar{m}_i$ for all i. Given a strategy $\sigma : FPath_{S(\bar{n})} \to Dist(Act_{\bar{n}})$ in $\mathcal{S}(\bar{n})$, we define the equivalent strategy $\hat{\sigma} : FPath_{\hat{\mathcal{S}}(\bar{m})} \to Dist(\hat{Act}_{\bar{m}})$ by:*

$$\hat{\sigma}(\hat{a} \mid \hat{\rho}) \triangleq \frac{\sum_{\rho \in FPath_{\mathcal{S}(\bar{n})_\sigma} : \hat{\rho} \sim \rho} \sum_{a \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{a})} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho) \cdot \sigma(a \mid \rho)}{\sum_{\rho \in FPath_{\mathcal{S}(\bar{n})_\sigma} : \hat{\rho} \sim \rho} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho)}$$

Note that after having followed a path $\hat{\rho}$ in the abstract system, we are not sure which of the paths ρ with $\hat{\rho} \sim \rho$ we want to simulate. So, we consider all the paths we could wish to simulate, and weight our choice of action by which ones are more likely. Before reasoning about the behaviour of this strategy, we verify its validity.

Observation 4.2. *$\hat{\sigma} : FPath_{\hat{\mathcal{S}}(\bar{m})} \to Dist(\hat{Act}_{\bar{m}})$ is a valid strategy.*

Proof. First, suppose that $\hat{\sigma}(\hat{a} \mid \hat{\rho}) > 0$. Then, there is at least one $\rho \in FPath_{\mathcal{S}(\bar{n})_\sigma}$ and one $a \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{a})$ such that $\hat{\rho} \sim \rho$ and $\sigma(a \mid \rho) > 0$. Then it must be the case that $a \in P_{\bar{n}}(last(\rho))$ since σ is a valid strategy. It follows that $\hat{a} \in \hat{P}_{\bar{n}}(last(\hat{\rho}))$.

It remains to check that for any $\hat{\rho} \in FPath_{\hat{\mathcal{S}}(\bar{m})}$ we have that $\sum_{\hat{a} \in \hat{Act}_{\bar{m}}} \hat{\sigma}(\hat{a} \mid \hat{\rho}) = 1$. For this notice that:

$$\sum_{\rho \in FPath_{\mathcal{S}(\bar{n})_\sigma} : \hat{\rho} \sim \rho} \sum_{\hat{a} \in \hat{Act}_{\bar{m}}} \sum_{a \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{a})} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho) \cdot \sigma(a \mid \rho) = \sum_{\rho \in FPath_{\mathcal{S}(\bar{n})_\sigma} : \hat{\rho} \sim \rho} \sum_{a \in Act_{\bar{n}}} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho) \cdot \sigma(a \mid \rho)$$

$$= \sum_{\rho \in FPath_{\mathcal{S}(\bar{n})_\sigma} : \hat{\rho} \sim \rho} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho)$$

with the first equality following simply from $\lambda_{\bar{n},\bar{m}}$ being a function and the second from σ being a valid strategy. So $\sum_{\hat{a} \in \hat{Act}_{\bar{m}}} \hat{\sigma}(\hat{a} \mid \hat{\rho}) = 1$, as desired. □

Our next lemma shows that the strategy we defined achieves the desired result – the probability of a path in the abstract model is equal to the sum of the probabilities of the concrete paths it represents.

Lemma 4.2. *Let $\bar{n}, \bar{m} \in \mathbb{N}_1^k$ with $\bar{n}_i > \bar{m}_i$ for all i. Let $\sigma : FPath_{S(\bar{n})} \to Dist(Act_{\bar{n}})$ be a strategy in $\mathcal{S}(\bar{n})$. Then it is the case that, for any path $\hat{\rho} \in FPath_{\hat{\mathcal{S}}(\bar{m})}$:*

$$\mathbf{P}_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}}(\hat{\rho}) = \sum_{\rho \in FPath_{\mathcal{S}(\bar{n})_\sigma} : \hat{\rho} \sim \rho} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho)$$

Proof. We proceed by induction on the length of the path $\hat{\rho}$. The base case is clear, as on both sides there is a unique path of length 1 with just the initial state, and this has probability 1.

Now, suppose the statement holds for a path $\hat{\rho}'$ and consider its extension $\hat{\rho} = \hat{\rho}'\hat{a}\hat{s}$ for some $\hat{a} \in \hat{Act}_{\bar{m}}$ and $\hat{s} \in \hat{S}_{\bar{m}}$. Note that if $\hat{s}.abs \neq \mathcal{R}_{last(\hat{\rho}').abs,\hat{a}}$ then both sides of the equation

are 0 as the sum on the right is empty, so we can ignore this case. Otherwise:

$$\begin{aligned}
\mathbf{P}_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}}(\hat{\rho}) &= \mathbf{P}_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}}(\hat{\rho}') \cdot \hat{\sigma}(\hat{a} \mid \hat{\rho}') \cdot \hat{t}_{\bar{m}}(\hat{s} \mid last(\hat{\rho}'), \hat{a}) \\
&= \left(\sum_{\rho' \in FPath_{\mathcal{S}(\bar{n})_\sigma} : \hat{\rho}' \sim \rho'} \sum_{a \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{a})} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho') \cdot \sigma(a \mid \rho') \cdot \hat{t}_{\bar{m}}(\hat{s} \mid last(\hat{\rho}'), \hat{a}) \right) \\
&= \sum_{\rho' \in FPath_{\mathcal{S}(\bar{n})_\sigma} : \hat{\rho}' \sim \rho'} \sum_{a \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{a})} \left(\mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho') \cdot \sigma(a \mid \rho') \cdot \sum_{g' \in S_{\bar{n}} : g' =_{conc} \hat{g}'} t_{\bar{n}}(g' \mid last(\rho'), a) \right) \\
&= \sum_{\rho \in FPath_{\mathcal{S}(\bar{n})_\sigma} : \hat{\rho} \sim \rho} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho)
\end{aligned}$$

with the first equality following from the definitions, the second from the inductive hypothesis, the third from Lemma 4.1 and the final one simply by rearranging. Thus, by induction on the length of the path, the lemma holds for all finite paths. □

This lemma completes our goal for this subsection and proves that it is possible for an abstract system to simulate the behaviour of larger concrete systems. This enables us to prove our main result of this subsection, which shows that if we can check a formula $P^{\max}_{\leq x}[\psi]$ on the abstract model, it will hold on larger models. We do this below.

Theorem 4.1. *Suppose $\hat{\mathcal{S}}(\bar{m}) \models P^{max}_{\leq x}[\psi]$ for some $\bar{m}$-indexed $PLTL_k$ formula. Then, $\mathcal{S}(\bar{n}) \models P^{max}_{\leq x}[\psi]$ for all $\bar{n} \in \mathbb{N}^k_1$ with $\bar{n}_i > \bar{m}_i$ for all i.*

Proof. We prove the contrapositive of this statement. Suppose we have $\mathcal{S}(\bar{n}) \models P^{\max}_{>x}[\psi]$ for some $\bar{n} \in \mathbb{N}^k_1$ with $\bar{n}_i > \bar{m}_i$ for all i. Then, by definition, for some strategy σ in $\mathcal{S}(\bar{n})$ we have:

$$\mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\{\rho \in IPath_{\mathcal{S}(\bar{n})_\sigma} : \rho \models \psi\}) > x$$

Now, let $\hat{\sigma}$ be the equivalent strategy in $\hat{\mathcal{S}}(\bar{m})$ as given by Definition 4.7. But now every path in the concrete model has a corresponding path in the abstract model where, by definition, the same atomic propositions hold at each step (and thus the same LTL formulas hold along the path). So:

$$\mathbf{P}_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}}(\{\hat{\rho} \in IPath_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}} : \hat{\rho} \models \psi\}) > x$$

since by Lemma 4.2 the probability of a path in the abstract model is the same as that of the corresponding paths in the concrete model.

Thus, by considering $\hat{\sigma}$ as our strategy, we see that $\hat{\mathcal{S}}(\bar{m}) \models P^{\max}_{>x}[\psi]$ and our result holds. □

Note that the above theorem gives a partial decision procedure for the parameterised model checking problem given in Definition 4.1. In particular, given an $\bar{m}$-indexed formula $P^{\max}_{\leq x}[\psi]$, if we can verify it on the abstract model $\hat{\mathcal{S}}(\bar{m})$ we have succeeded in checking it for all larger systems. However, if wish to have a complete decision procedure we also need to develop a converse result. This is what the next subsection aims to do.

4.2.2 Simulating the Abstract System

In order to develop a converse to the theorem in the previous subsection, we need to prove that large concrete systems simulate the behaviour of the abstract system. While they will not be able to exactly simulate the abstract system, we will see that they can do so to an arbitrarily high probability for paths of a finite length. Because of the way our logic PLTL_{k} is defined, this will be sufficient to obtain a converse to Theorem 4.1. Before we can do this, though, we need to give a few technical definitions. Our first definition describes the paths that an individual agent *could* follow with some non-zero probability.

Definition 4.8 (Finite Local Path)**.** *Let $\mathcal{S}$ be an SPMAS with k agent templates, and $l \in \mathbb{N}_1$ be a path length. We say that $\rho = s_1 a_1 s_2 \dots s_l$ where each $s_j \in S_i$ and each $a_j \in Act_i$ is a valid finite local path if:*

(i) For all $1 \leq j \leq l$, $a_j \in P_i(s_j)$.

(ii) For all $1 \leq j < l$, there is some $X \subseteq \cup_{j \in k} Act_j$ and some $a_E \in Act_E$ such that $t_i(s_{j+1} \mid s_j, a_E, X, a_j) > 0$. We write p_j for the least such non-zero value of $t_i(s_{j+1} \mid s_j, a_E, X, a_j)$.

If ρ is a valid finite local path, we write $MinProb(\rho) = \prod_{j=1}^{l-1} p_j$.

Notice that $MinProb(\rho)$ gives us a lower bound on the probability of a given finite local path, assuming that the other agents and the environment are behaving in such a way as to minimise our agent's chances of managing to follow the local path ρ.

We denote the set of all finite local paths of length l for agents of type i by $Paths_{l,i}$. Note that $Paths_{l,i}$ is a finite set, so we can enumerate its elements. Whenever we do this, we will assume this is always done in the same ordering so that the r-th element used in one place is the same as the r-th element elsewhere.

Now, we aim to define a number of agents $\bar{N}_i^{E,l}$ for each type of agent i, such that should our strategy wish to do so, it can ensure that for any path of length l the probability of at least one agent performing every possible transition is at least E. Note that since the transitions of agents are independent, one sufficient number for this to be the case is to have enough agents following each path that the probability of them succeeding is at least $E^{\frac{1}{kr_i}}$ where r_i is the number of different local paths for agents of type i. This is the idea underpinning the definition below.

Definition 4.9 (Sufficient Number)**.** *Let $\mathcal{S}$ be an SPMAS with k different types of agents, $l \in \mathbb{N}_1$ a path length and $0 < E < 1$ a target probability. Further, let $Paths_{l,i} = \{\rho_{i,1}, \ldots, \rho_{i,r_i}\}$ for each agent type i. Now, we define the sufficient number of agents of type i to ensure with probability E that the first r paths are followed by:*

$$s_i(E, l, r) = \sum_{1 \leq j \leq r} \left\lceil log_{1-MinProb(\rho_{i,j})}(1 - E^{\frac{1}{kr_i}}) \right\rceil$$

Finally, we define the sufficient number $\bar{N}^{E,l} \in \mathbb{N}_1^k$ by $\bar{N}_i^{E,l} = s_i(E, l, r_i)$.

Certainly, tighter choices of sufficient numbers are possible by exploiting the fact that many finite paths will share the same prefixes. However, as our bound $\bar{N}^{E,l}$ is only used in theoretical results and does not affect the running time of any of our algorithms (indeed, it does not ever even need explicitly calculating), the above is adequate for our purposes.

Before we continue, we give a definition of how we can map a path in the concrete model to a corresponding one in the abstract model. While similar, this definition is stronger than our previous notion in Definition 4.6 of an abstract path simulating a concrete one.

Definition 4.10 (Path Abstraction)**.** *Let $\bar{n}, \bar{m} \in \mathbb{N}_1^k$ with $\bar{n}_i > \bar{m}_i$ for all i. We define the path abstraction map $\lambda_{\bar{n},\bar{m}} : IPath_{\mathcal{S}(\bar{n})} \to IPath_{\hat{\mathcal{S}}(\bar{m})}$ by mapping each abstract path $\rho = s_1 a_1 s_2 \ldots$ to the unique abstract path $\hat{\rho} = \hat{s}_1 \hat{a}_1 \hat{s}_2 \ldots$ such that for all $i \in \mathbb{N}_1$:*

(i) $\hat{s}_i = \lambda_{\bar{n},\bar{m}}(s_i)$

(ii) $\hat{a}_i = \lambda_{\bar{n},\bar{m}}(a_i)$

Like before, we restrict this definition to finite paths in the obvious way. Equipped with this notion, we can now go on to define the strategy by which a concrete system of size $\bar{m} + \bar{N}^{E,l}$ can simulate paths of length up to l in the abstract system of size $\bar{m}$ with a probability E of success. We do this below.

Definition 4.11 (Simulating Strategy)**.** *Let $\bar{m} \in \mathbb{N}_1^k$, $l \in \mathbb{N}_1$, $0 < E < 1$ and $\hat{\sigma} : FPath_{\hat{\mathcal{S}}(\bar{m})} \to \hat{Act}_{\bar{m}}$. As before, let $Paths_{l,i} = \{\rho_{i,1}, \ldots, \rho_{i,r_i}\}$ and let $\rho_{i,r} = s_{i,r,1} a_{i,r,1} \ldots s_{i,r,l}$. Then, we define the simulating strategy $\sigma_{E,l} : FPath_{\mathcal{S}(\bar{m}+\bar{N}^{E,l})} \to Act_{\bar{m}+\bar{N}^{E,l}}$ by:*

$$\sigma_{E,l}(\rho).(i,j) \triangleq$$

$$\begin{cases} \hat{\sigma}(\lambda_{\bar{m}+\bar{N}^{E,l},\bar{m}}(\rho)).(i,j) & \text{if } 1 \leq j \leq \bar{m}_i \\ a_{i,r,o} & \text{if } \mid \rho \mid = o < l \text{ and for some } 1 \leq r \leq r_i \\ & \qquad \bar{m}_j + s_i(E,l,r-1) < j \leq \bar{m}_j + s_i(E,l,r) \\ & \qquad \text{and } a_{i,r,o} \in P_i(last(\rho).(i,j)) \cap \hat{\sigma}(\lambda_{\bar{m}+\bar{N}^{E,l},\bar{m}}(\rho)).abs \\ \varepsilon & \text{otherwise} \end{cases}$$

In our simulating strategy, the first $\bar{m}$ agents follow the same behaviour as the ones in the abstract system. Then, for each type of agent i and each possible finite path $\rho_{i,j}$ we have

$\lceil log_{1-MinProb(\rho_{i,r})} \rceil$ agents attempt to follow that path if doing so respects the strategy of the abstract system (otherwise, they just perform the null action ε). Before going on to prove a lemma about the behaviour of this strategy, we check its validity.

Observation 4.3. *$\sigma_{E,l} : FPath_{\mathcal{S}(\bar{m}+\bar{N}^{E,l})} \to Act_{\bar{m}+\bar{N}^{E,l}}$ is a valid strategy.*

Proof. We need to show that for any path $\rho \in FPath_{\mathcal{S}(\bar{m}+\bar{N}^{E,l})}$, and for every $(i,j) \in \mathcal{A}(\bar{m}+\bar{N}^{E,l})$ we have $\sigma_{E,l}(\rho).(i,j) \in P_i(last(\rho).(i,j))$. Note that our definition has three cases. In the first case, the result follows from $\hat{\sigma}$ being a valid strategy. In the second case, it is enforced in the condition for the case that $\sigma_{E,l}(\rho).(i,j) \in P_i(last(\rho).(i,j)$. Finally, in the last case note that the null action ε is always enabled. This covers all cases for our definition and completes the proof. □

Having proved the validity of our strategy, we note that following it will ensure with probability at least E that our simulating strategy manages to perform every transition encoded in the abstract system for the first l time steps. This notion is formalised in the lemma below.

Lemma 4.3. *Let $\bar{m} \in \mathbb{N}_1^k$, $l \in \mathbb{N}_1$, $0 < E < 1$ and $\hat{\sigma} : FPath_{\hat{\mathcal{S}}(\bar{m})} \to \hat{Act}_{\bar{m}}$. Then, for any path $\hat{\rho} \in FPath_{\hat{\mathcal{S}}(\bar{m})}$ with $\mid \hat{\rho} \mid = l$ it is the case that:*

$$\sum_{\rho \in \lambda^{-1}_{\bar{m}+\bar{N}^{E,l},\bar{m}}(\hat{\rho})} \mathbf{P}_{\mathcal{S}(\bar{m}+\bar{N}^{E,l})_{\sigma_{E,l}}}(\rho) \geq E \cdot \mathbf{P}_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}}(\hat{\rho})$$

Proof. Let $\hat{\rho} = \hat{s}_1 \hat{a}_1 \ldots \hat{s}_l$ and for each type of agent i take $Paths_{l,i} = \{\rho_{i,1}, \ldots, \rho_{i,r_i}\}$ to be its set of local paths. We write $\rho_{i,r} = s_{i,r,1} a_{i,r,1} \ldots s_{i,r,l}$. Given a concrete path $\rho = s_1 a_1 \ldots s_l$ we the j-th agent of type i followed path $\rho_{i,r}$ if for all $1 \leq o \leq l$ we have $s_o.(i,j) = s_{i,r,o}$ and for all $1 \leq o < l$ we have $a_o.(i,j) = a_{i,r,o}$. When this is the case, we write $\rho \mid_{i,j} = \rho_{i,r}$.

We say a local path $\rho_{i,r}$ is *enabled* if for all $1 \leq o < l$, $a_{i,r,o} \in \hat{a}_o.abs$. We let $EPaths_{l,i} \subseteq Paths_{l,i}$ be the subset of local paths that are enabled.

Now, consider the following set of paths:

$$X = \{\rho \in \lambda^{-1}_{\bar{m}+\bar{N}^{E,l},\bar{m}}(\hat{\rho}) : \forall i \in \dot{k} \forall \rho_{i,r} \in EPaths_{l,i} \text{ there is some } j \text{ with } \bar{m}_j + s_i(E,l,r-1) < j \leq \bar{m}_j + s_i(E,l,r) \text{ such that } \rho\,|_{i,j} = \rho_{i,r}\}$$

Since $X \subseteq \lambda^{-1}_{\bar{m}+\bar{N}^{E,l},\bar{m}}(\hat{\rho})$, to show our result it would be sufficient to show:

$$\sum_{\rho \in X} \mathbf{P}_{\mathcal{S}(\bar{n})_{\sigma_{E,l}}}(\rho) \geq E \cdot \mathbf{P}_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}}(\hat{\rho})$$

In particular, our result will hold if we show that the probability in $\mathcal{S}(\bar{n})_{\sigma_{E,l}}(\rho)$ of at least one of the abstract agents following each path in $EPaths_{l,i}$ is at least E. We note that for each $\rho_{i,r} \in EPaths_{l,i}$ there are

$$m_{i,r} = \left\lceil log_{1-MinProb(\rho_{i,r})}(1 - E^{\frac{1}{kr_i}}) \right\rceil$$

agents attempting to follow it. It follows that the probability that at least one succeeds is at least $1 - (1 - MinProb(\rho_{i,j}))^{m_{i,r}}$. Now, the probability that this is the case for all types of agent i and all paths $\rho_{i,r} \in EPaths_{l,i}$ is at least:

$$\prod_{i=1}^{k} \prod_{\rho_{i,r} \in EPaths_{l,i}} (1 - (1 - MinProb(\rho_{i,r}))^{m_{i,r}}) \geq \prod_{i=1}^{k} \prod_{\rho_{i,r} \in EPaths_{l,i}} E^{\frac{1}{kr_i}} \geq E$$

by applying the definitions and then noting that for each i there are at most r_i paths in $EPaths_{l,i}$. This completes our proof. □

This lemma proves that large concrete systems can simulate the behaviour of the abstract system for a finite number of steps to an arbitrarily high probability. Having proved this, we are now equipped to show the main result of this subsection which is a converse of Theorem 4.1.

Theorem 4.2. *Suppose $\hat{\mathcal{S}}(\bar{m}) \not\models P^{max}_{\leq x}[\psi]$ for some $\bar{m}$-indexed PLTL$_k$ formula ψ. Then, $\mathcal{S}(\bar{n}) \not\models P^{max}_{\leq x}[\psi]$ for some $\bar{n} \in \mathbb{N}^k_1$ with $\bar{n}_i > \bar{m}_i$ for all i.*

Algorithm 4.1 Decision procedure for the PMCP of SPMAS against PLTL_{k}

Input: SPMAS $\mathcal{S}$, PLTL_{k} formula ϕ

Output: Boolean

1: **function** PMCP-BOUNDED($\mathcal{S}$, ϕ)
2: $\bar{m} \leftarrow$ ComputeIndex(ϕ)
3: Build the abstract model $\hat{\mathcal{S}}(\bar{m})$ using Definition 4.2
4: **return** $\hat{\mathcal{S}}(\bar{m}) \models \phi$
5: **end function**

Proof. If our assumption holds, then there is a strategy $\hat{\sigma} : \hat{S}_{\bar{m}} \to \hat{Act}_{\bar{m}}$ such that

$$P_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}}(\{\hat{\rho} \in IPath_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}} : \hat{\rho} \models \psi\}) > x > 0 \tag{4.1}$$

Let $l = tb(\psi)$ be the time bound of the path formula ψ. Then, we define $\bar{n} = \bar{m} + \bar{N}^{E,l}$ and take $\sigma_{E,l}$ to be the equivalent strategy for $0 < E < 1$. By Lemma 4.3:

$$\mathbf{P}_{\mathcal{S}(\bar{n})_{\sigma_{E,l}}}(\{\hat{\rho} \in IPath_{\mathcal{S}(\bar{n})_{\sigma_{E,l}}} : \rho \models \psi\}) \geq E \cdot \mathbf{P}_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}}(\{\hat{\rho} \in IPath_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}} : \hat{\rho} \models \psi\})$$

So, the result holds by simply taking:

$$E = \frac{x + \mathbf{P}_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}}(\{\hat{\rho} \in IPath_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}} : \hat{\rho} \models \psi\})}{2 \cdot \mathbf{P}_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}}(\{\hat{\rho} \in IPath_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}} : \hat{\rho} \models \psi\})}$$

noting $0 < E < 1$ by Equation (4.1). □

Note that together with Theorem 4.1, this gives a complete decision procedure for the PMCP in Definition 4.1. In particular, to check the validity of an $\bar{m}$-indexed formula on arbitrarily large systems, we just need to check its validity in the abstract system $\hat{\mathcal{S}}(\bar{m})$. We formalise this procedure and show its correctness in the next section.

4.3 Verification Procedure

In this section, we tie together the results from the rest of this chapter, showing that we have developed a technique to solve the parameterised model checking problem of SPMAS

against PLTL_k specifications. This simple procedure is shown in Algorithm 4.1. We now show its correctness using the results from the previous section.

Corollary 4.1. *`PMCP-Bounded` is a sound and complete decision procedure for the PMCP in Definition 4.1.*

Proof. In order to prove soundness, we need to show that `PMCP-Bounded`$(\mathcal{S}, \phi)$ holds iff $\mathcal{S} \models \phi$. When the return value is *true*, this follows from Theorem 4.1. For return values of *false* it follows from Theorem 4.2. Completeness is clear as there are no loops. □

Having shown the correctness of our procedure, we make an observations regarding its time complexity.

Observation 4.4. *The run-time of `PMCP-Bounded`$(\mathcal{S}, \phi)$ is exponential in the number of states in agent templates, polynomial in the number of environment states, and doubly exponential in the size of the formula.*

Proof. First, observe that the states of $\hat{\mathcal{S}}(\bar{m})$ are drawn from $S_{\bar{m}} \times \mathcal{P}(\cup_{i=1}^{k} S_i)$ and therefore the size of $\hat{\mathcal{S}}(\bar{m})$ is exponential in $\mid \cup_{i=1}^{k} S_i \mid$ but polynomial in $\mid S_E \mid$. Further, note that our underlying procedure for checking $\hat{\mathcal{S}}(\bar{m}) \models \phi$ is polynomial in the size of the MDP and doubly exponential in the size of the formula (see Section 7 of [Forejt et al., 2011]). This gives our desired result. □

In practice, this complexity result will mean that, as we see in Chapter 7, our method is limited to verifying properties of systems where the number of different states agents can be in is small.

4.4 Summary

In this chapter, we have developed a technique to solve the PMCP of SPMAS against PLTL_k specifications (Definition 4.1). To achieve this, we first defined an abstract model for SPMAS (Definition 4.2). We then proved both that the abstract model can simulate larger systems (Theorem 4.1) and conversely that larger systems can simulate the abstract system (Theorem 4.2). These two results were then combined into a verification procedure (Algorithm 4.1) which we proved to be sound and complete (Corollary 4.1).

Chapter Five

Verifying Unbounded Properties

In this chapter, we will define the parameterised model checking problem (PMCP) for APMAS against PLTL specifications. Similarly to the previous chapter, we will present an abstract model which allows us to solve this decision problem. This model will follow the same inspiration as the one in the previous chapter, but adapted to APMAS rather than SPMAS. As in the previous chapter, we will introduce a decision procedure using our abstract model, and show that it is sound. The procedure presented here will not, however, be complete as the richer decision problem considered here is undecidable in general.

The material presented in this chapter has appeared, in a shorter form, in our 2019 AAMAS paper [Lomuscio and Pirovano, 2019].

5.1 Parameterised Model Checking Problem

In this section we define the parametrised model checking problem that we will address this chapter. In particular, this is the problem of checking whether a PLTL formula holds in an APMAS of any size. Notice that while this problem appears similar to the one in Definition 4.1, because the underlying models and specifications are different, distinct techniques are required.

Definition 5.1 (PMCP of PLTL on APMAS)**.** *Given an APMAS $\mathcal{S}$ and an $\bar{m}$-indexed*

PLTL formula ϕ, the parameterised model checking problem (PMCP) involves establishing whether it is the case that $\mathcal{S}(\bar{n}) \models \phi$ for all $\bar{n}$ with $\bar{n}_i > \bar{m}_i$ for all i. We write $\mathcal{S} \models \phi$ if this is the case.

Since this is a probabilistic extension of a problem that is known to be undecidable in general [Apt and Kozen, 1986], it is clear that it is also undecidable in general. The rest of this chapter is devoted to building a sound but incomplete procedure for solving the above problem using an abstract model.

5.2 Abstract Model

As in the last chapter, our abstract model will aim to capture the behaviour of arbitrarily large concrete systems. This will mean that for every path in a concrete system (of any size) there will be a corresponding path in our abstract model. We will exploit this property to give a partial decision procedure for the parameterised model checking problem.

We now go on to define our abstract model which is again based on a counter abstraction technique [Pnueli et al., 2002].

Definition 5.2 (Abstract Model of an APMAS)**.** *Given an APMAS $\mathcal{S}$ and an $\bar{m} \in \mathbb{N}_1^k$, the abstract model of $\bar{m}$ agents is defined by $\hat{\mathcal{S}}(\bar{m}) = \langle \hat{S}_{\bar{m}}, \hat{\iota}_{\bar{m}}, \hat{Act}_{\bar{m}}, \hat{P}_{\bar{m}}, \hat{t}_{\bar{m}}, \hat{V}_{\bar{m}} \rangle$, where:*

- $\hat{S}_{\bar{m}} \triangleq S_{\bar{m}} \times \mathcal{P}(\bigcup_{i=1}^{k} S_i)$ *is the set of abstract global states.*
- $\hat{\iota}_{\bar{m}} \triangleq (\iota_{\bar{m}}, \{\iota_1, \ldots, \iota_k\})$ *is the initial abstract global state.*
- $\hat{Act}_{\bar{m}} \triangleq Act_{\bar{m}} \cup \bigcup_{i=1}^{k}((A_i \cup AE_i) \times S_i \times \{\uparrow, \downarrow\})$ *is the set of abstract global actions.*

- $\hat{P}_{\bar{m}} : \hat{S}_{\bar{m}} \to \mathcal{P}(\hat{Act}_{\bar{m}})$ *is defined by:*

$$\hat{P}_{\bar{m}}(g, X) \triangleq (P_{\bar{m}}(g) \setminus \{a \in GS : \exists i \in \dot{k}, x \in X \text{ with } x \in S_i \text{ and } a \notin P_i(x)\}) \cup \bigcup_{i=1}^{k} \{(a, s, v) \in A_i \times X \times \{\uparrow, \downarrow\} : a \in P_i(s)\} \cup \bigcup_{i=1}^{k} \{(a, s, v) \in AE_i \times X \times \{\uparrow, \downarrow\} : a \in P_i(s) \cap P_E(g.E)\}$$

- $\hat{t}_{\bar{m}} : \hat{S}_{\bar{m}} \times \hat{Act}_{\bar{m}} \to Dist(\hat{S}_{\bar{m}})$ *is the transition function, defined for asynchronous agent actions by:*

$$\hat{t}_{\bar{m}}((g', X') \mid (g, X), a, l, \uparrow)) \triangleq \begin{cases} t_i(l' \mid l, a) & \text{if } X' \setminus X = \{l'\} \text{ and } l, l' \in S_i \text{ and } g = g' \\ \sum_{l' \in X \cap S_i} t_i(l' \mid l, a) & \text{if } X' = X \text{ and } l \in S_i \text{ and } g = g' \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{t}_{\bar{m}}((g', X') \mid (g, X), (a, l, \downarrow)) \triangleq \begin{cases} t_i(l' \mid l, a) & \text{if } X' = (X \setminus \{l\}) \cup \{l'\} \text{ and } l \in S_i \\ & \quad \text{and } l' \in S_i \setminus X \text{ and } g = g' \\ \sum_{l' \in X' \cap S_i} t_i(l' \mid l, a) & \text{if } X' = X \setminus \{l\} \text{ and } l \in S_i \text{ and } g = g' \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{t}_{\bar{m}}((g', X') \mid (g, X), (a, i)) \triangleq \begin{cases} t_{\bar{m}}(g' \mid g, a) & \text{if } X' = X \\ 0 & \text{otherwise} \end{cases}$$

For agent-environment actions, we similarly define:

$$\hat{t}_{\bar{m}}((g', X') \mid (g, X), (a, l, \uparrow)) \triangleq t_E(g'.E \mid g.E, a) \times$$
$$\begin{cases} t_k(l' \mid l, a) & \text{if } X' \setminus X = \{l'\} \text{ and } l, l' \in S_k \\ & \quad \text{and } \forall (i,j) \in \mathcal{A}(\bar{m}) : g.(i,j) = g'.(i,j) \\ \sum_{l' \in X \cap S_k} t_k(l' \mid l, a) & \text{if } X' = X \text{ and } l \in S_k \\ & \quad \text{and } \forall (i,j) \in \mathcal{A}(\bar{m}) : g.(i,j) = g'.(i,j) \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{t}_{\bar{m}}((g', X') \mid (g, X), (a, l, \downarrow)) \triangleq t_E(g'.E \mid g.E, a) \times$$
$$\begin{cases} t_k(l' \mid l, a) & \text{if } X' = (X \setminus \{l\}) \cup \{l'\} \text{ and } l \in S_k \text{ and } l' \in S_k \setminus X \\ & \quad \text{and } \forall (i,j) \in \mathcal{A}(\bar{m}) : g.(i,j) = g'.(i,j) \\ \sum_{l' \in X' \cap S_k} t_k(l' \mid l, a) & \text{if } X' = X \setminus \{l\} \text{ and } l \in S_k \\ & \quad \text{and } \forall (i,j) \in \mathcal{A}(\bar{m}) : g.(i,j) = g'.(i,j) \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{t}_{\bar{m}}((g', X') \mid (g, X), (a, i)) \triangleq \begin{cases} t_{\bar{m}}(g' \mid g, a) & \text{if } X' = X \\ 0 & \text{otherwise} \end{cases}$$

For asynchronous environment actions, we define:

$$\hat{t}_{\bar{m}}((g', X') \mid (g,X), a) \triangleq \begin{cases} t_{\bar{m}}(g' \mid g, a) & \text{if } X' = X \\ 0 & \text{otherwise} \end{cases}$$

Finally, for global actions we define:

$$\hat{t}_{\bar{m}}((g', X') \mid (g, X), a) \triangleq$$
$$\begin{cases} t_{\bar{m}}(g' \mid g, a) & \text{if } X' = \{s' \in S \mid \exists i \in \dot{k}, s \in X \cap S_i : t_i(s' \mid s, a) = 1\} \\ 0 & \text{otherwise} \end{cases}$$

- $\hat{V}_{\bar{m}} : \hat{S}_{\bar{m}} \to \mathcal{P}((AP \times \mathcal{A}(\bar{m})) \cup AP)$ *is the labelling function given by* $\hat{V}_{\bar{m}}(g, X) \triangleq V_{\bar{m}}(g)$.

Intuitively, the abstract global state records the exact state of the first $\bar{m}$ agents (since we need this to check whether a formula holds). For all other agents, it stores a set of states which corresponds to the states that have *one or more* agents in that state. We will refer to these agents in the second component of the state as the *abstract agents*.

Abstract global actions are either concrete actions of the first $\bar{m}$ agents, or they are actions of the abstract agents. In the latter case, we add some further labelling to the actions that we need to resolve how the transition occurs. In particular, we label the action with both the state it was performed from and whether the agent performing the action was the only agent in that state ("shrinking" actions, labelled by $\downarrow$) or one of several ("growing" actions, labelled by $\uparrow$).

The protocol then enables all concrete actions for the concrete agents with the exception of global-synchronous actions that cannot be performed by one of the abstract agents. For the abstract agents, it enables their actions along with the corresponding labelling (notice that since we do not actually know how many agents are in each state, we always enable both the $\uparrow$ and the $\downarrow$ version of actions).

For the labelling function, we are only interested in the $\bar{m}$ concrete agents in the first component of the state (since these are the ones for which corresponding atomic propositions are used in the formula), so we simply discard the second component.

The first transitions of an example abstract system can be seen in Figure 5.1. In the initial state, the two concrete agents $(1, 1)$ and $(2, 1)$ may perform their concrete transitions with a and e actions exactly as in Figure 3.4. Further, the abstract agents can also perform a or e. This gives abstract actions like $(a, 1, \uparrow)$ which denotes one of several abstract agents in state 1 performing the a action and could result in a transition that updates the abstract state from $\{1, 3\}$ to $\{1, 2, 3\}$. The abstract action $(a, 1, \downarrow)$ corresponds to the last abstract agent in state 1 performing the a action and may result in the abstract state being updated

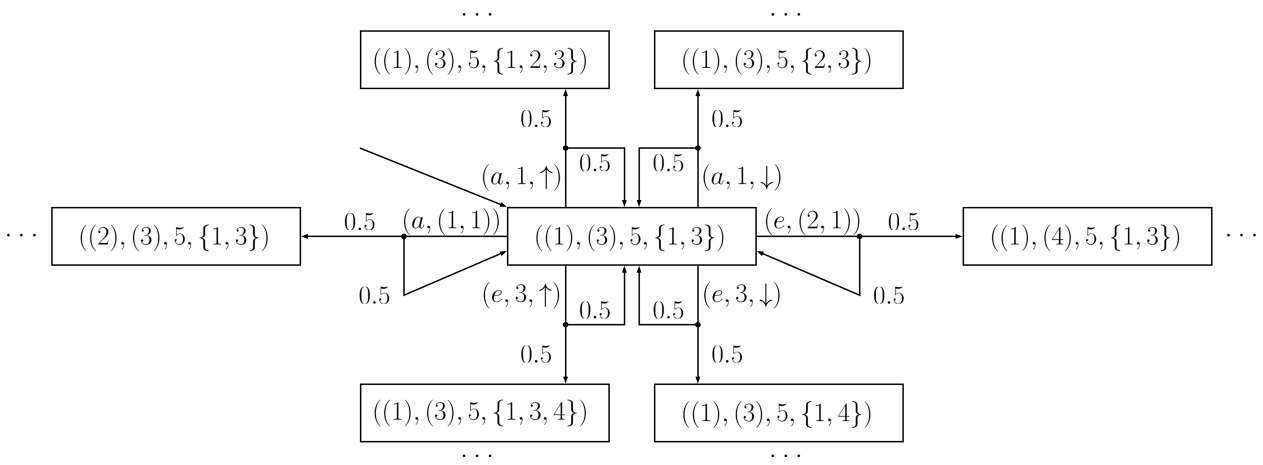


Figure 5.1 The first state and its outgoing transitions of the abstract system $\hat{\mathcal{S}}((1,1))$ where $\mathcal{S}$ is the APMAS from Figure 3.3. Note the full abstract system is much larger (it contains 19 states and 105 transitions) so is not shown.

from $\{1,3\}$ to $\{2,3\}$. The transitions for the abstract e action are similar.

Before we proceed, it remains to check that the transition function we have defined gives a valid probability distribution. We do so now.

Observation 5.1. *For any $s \in \hat{S}_{\bar{m}}$ and $a \in \hat{Act}_{\bar{m}}$, it is the case that:*

$$\sum_{s' \in \hat{S}_{\bar{m}}} \hat{t}_{\bar{m}}(s' \mid s, a) = 1$$

Proof. Let $\hat{s} = (g, X) \in \hat{S}_{\bar{m}}$. We consider different choices of action.

- $a \in Act_{\bar{m}} \setminus GS$. Then:

$$\sum_{s' \in \hat{S}_{\bar{m}}} \hat{t}_{\bar{m}}(s' \mid (g, X), a) = \sum_{g' \in S_{\bar{m}}} \hat{t}_{\bar{m}}((g', X) \mid (g, X), a) = \sum_{g' \in S_{\bar{m}}} t_{\bar{m}}(g' \mid g, a) = 1$$

  The equalities follow by noting which terms are non-zero, then applying the definition of $\hat{t}$ and finally using Observation 3.2.

- $a \in GS$. Similarly to the above case, we have:

$$\sum_{s' \in \hat{S}_{\bar{m}}} \hat{t}_{\bar{m}}(s' \mid (g, X), a) = \sum_{g' \in S_{\bar{m}}} \hat{t}_{\bar{m}}((g', X') \mid (g, X), a) = \sum_{g' \in S_{\bar{m}}} t_{\bar{m}}(g' \mid g, a) = 1$$

where $X' = \{s' \in S \mid \exists i \in \dot{k}, s \in X \cap S_i : t_i(s' \mid s, a) = 1\}$. Once again, this follows by picking out the non-zero terms of the sum, applying the definition of $\hat{t}$, and finally using Observation 3.2.

- $a = (a', l, \uparrow)$ for $a' \in A_i$ and $l \in S_i$. Then:

$$\sum_{s' \in \hat{S}_{\bar{m}}} \hat{t}_{\bar{m}}(s' \mid (g, X), a) = \sum_{l' \in S_i \setminus X} t_i(l' \mid l, a) + \sum_{l' \in X \cap S_i} t_i(l' \mid l, a) = \sum_{l' \in S_i} t_i(l' \mid l, a) = 1$$

by picking out the non-zero terms, performing some rearranging and then noting that t_i is a valid transition function.

- $a = (a', l, \downarrow)$ for $a' \in A_i$ and $l \in S_i$. This is almost identical to the above case.

- $a = (a', l, \uparrow)$ for $a' \in AE_i$ and $l \in S_i$. Then:

$$\begin{aligned}\sum_{s' \in \hat{S}_{\bar{m}}} \hat{t}_{\bar{m}}(s' \mid (g, X), a) &= \sum_{s_E \in S_E} \Bigg(\sum_{l' \in S_i \setminus X} t_E(s_E \mid g.E, a') t_i(l' \mid l, a) \\ &\qquad + \sum_{l' \in X \cap S_i} t_E(s_E \mid g.E, a') t_i(l' \mid l, a) \Bigg) \\ &= \sum_{s_E \in S_E} \left(t_E(s_E \mid g.E, a') \sum_{l' \in S_i} t_i(l' \mid l, a) \right) \\ &= \sum_{s_E \in S_E} t_E(s_E \mid g.E, a') = 1\end{aligned}$$

by picking out the non-zero terms, performing some rearranging and then noting first that t_i is a valid transition function and subsequently that t_E is also a valid transition function.

- $a = (a', l, \uparrow)$ for $a' \in AE_i$ and $l \in S_i$. This is almost identical to the above case.

This covers all possible cases for the action a and completes our proof. □

Having defined our abstract system and shown that this definition is correct, we now wish to formalise in what sense this captures the same paths as concrete systems. Firstly, we borrow the concept on a *state abstraction map* $\lambda_{\bar{n},\bar{m}} : S_{\bar{n}} \to \hat{S}_{\bar{m}}$ exactly as in Definition 4.3

(with SPMAS replaced by APMAS). However, as the actions of agents in APMAS are distinct from those in SPMAS we will need a new notion of action abstraction. We define this below.

Definition 5.3 (Action Abstraction)**.** *Let $\bar{n}, \bar{m} \in \mathbb{N}_1^k$ with $\bar{n}_i > \bar{m}_i$ for all i. Then, the abstraction map on actions $\lambda_{\bar{n},\bar{m}} : S_{\bar{n}} \times Act_{\bar{n}} \to \hat{Act}_{\bar{m}}$ is given by:*

$$\lambda_{\bar{n},\bar{m}}(\bar{s}, a) \triangleq \begin{cases} a & \text{if } a \in Act_{\bar{m}} \\ (a', \bar{s}.(i,j), \downarrow) & \text{if } a = (a', (i,j)) \text{ for some } a' \in (A_i \cup AE_i) \text{ and} \\ & \quad (i,j) \in \mathcal{A}(\bar{n}) \setminus \mathcal{A}(\bar{m}) \text{ with} \\ & \quad \mid \{j' \in \{\bar{m}_i + 1, \ldots, \bar{n}_i\} \mid \bar{s}.(i,j) = \bar{s}.(i,j')\} \mid = 1 \\ (a', \bar{s}.(i,j), \uparrow) & \text{if } a = (a', (i,j)) \text{ for some } a' \in (A_i \cup AE_i) \text{ and} \\ & \quad (i,j) \in \mathcal{A}(\bar{n}) \setminus \mathcal{A}(\bar{m}) \text{ with} \\ & \quad \mid \{j' \in \{\bar{m}_i + 1, \ldots, \bar{n}_i\} \mid \bar{s}.(i,j) = \bar{s}.(i,j')\} \mid > 1 \end{cases}$$

Intuitively, for each type of agent i this maps actions concerning the first $\bar{m}_i$ such agents to themselves. For actions of the remaining agents, these are labelled with the state that they occurred from and whether they were shrinking ($\downarrow$) or growing ($\uparrow$) ones.

Note that unlike the similar Definition 4.4, here we also need the state the action is being performed from in order to map a concrete action to an abstract one as without this we would not be able to know whether we should use the shrinking or growing version of an action.

We now give a technical lemma that will be helpful to prove the the validity of our algorithm later. Intuitively, this lemma says that the probability of a transition in the abstract model is precisely the sum of the probabilities of the concrete transitions it could represent.

Lemma 5.1. *Let $\bar{n}, \bar{m} \in \mathbb{N}_1^k$ with $\bar{n}_i > \bar{m}_i$ for all i. Let $g \in S_{\bar{n}}$ and $a \in P_{\bar{n}}(g)$. Then, for*

any $\hat{g}' \in \hat{S}_{\bar{m}}$*:*

$$\hat{t}_{\bar{m}}(\lambda_{\bar{n},\bar{m}}(g,a) \mid \hat{g}', \lambda_{\bar{n},\bar{m}}(g)) = \sum_{g' \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{g}')} t_{\bar{n}}(g' \mid g, a)$$

Proof. Let $g = (s_1, \ldots, s_k, s_E)$ and $\lambda_{\bar{n},\bar{m}}(g) = ((\hat{s}_1, \ldots, \hat{s}_k, \hat{s}_E), X)$. Denote by $\hat{g}' = ((\hat{s}'_1, \ldots, \hat{s}'_k, \hat{s}'_E), X')$. We now consider different possible cases for a:

- $a \in A_E$. Note that if $X \neq X'$, both sides of the equation are 0 following the definitions. Similarly, if for any $i \in \mathbb{N}_1$ we have $\hat{s}_i \neq \hat{s}'_i$ then both sides are 0. It remains to consider the case where $X = X'$ and $\hat{s}_i = \hat{s}'_i$ for all $i \in \mathbb{N}_1$. Then, note that:

  $$\hat{t}_{\bar{m}}(\hat{g}' \mid \lambda_{\bar{n},\bar{m}}(g), \lambda_{\bar{n},\bar{m}}(g,a)) = t_{\bar{m}}((\hat{s}_1, \ldots, \hat{s}_k, \hat{s}'_E) \mid (\hat{s}_1, \ldots, \hat{s}_k, \hat{s}_E), a) = t_E(\hat{s}'_E \mid s_E, a)$$

  by following the definitions from the left hand side. On the right hand side we have:

  $$\sum_{g' \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{g}')} t_{\bar{n}}(g' \mid g, a) = t_{\bar{n}}((s_1, \ldots, s_k, \hat{s}'_E) \mid (s_1, \ldots, s_k, s_E), a) = t_E(\hat{s}'_E \mid s_E, a)$$

  with the first equality following from the fact that since $a \in A_E$, the only cases for $t_{\bar{n}}$ that are not 0 are those where only the state of the environment changes. The second equality follows by the definition giving the result.

- $a \in GS$. Note that since the agent transitions are deterministic (Definition 3.8), if we have $X' \neq \{s' \in S \mid \exists i \in \dot{k}, s \in X \cap S_i : t_i(s' \mid s, a) = 1\}$ then both sides would be equal to 0 (the left simply by definition, the right by noting that no $g' \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{g}')$ is reachable from g so every term being summed is 0). So, we need only consider the case when $X' = \{s' \in S \mid \exists i \in \dot{k}, s \in X \cap S_i : t_i(s' \mid s, a) = 1\}$.

  Further note that (once again by the agent transitions being deterministic) there are unique states $\hat{p}_1, \ldots, \hat{p}_k \in S_{\bar{m}}$ such that $t_{\bar{m}}((\hat{p}_1, \ldots, \hat{p}_k, \hat{s}'_E) \mid (\hat{s}_1, \ldots, \hat{s}_k, \hat{s}_E), a) \neq 0$. If for any $i \in \mathbb{N}_1$ it is the case that $\hat{s}'_i \neq \hat{p}_i$ then both sides are equal to 0. So, we only need to check that case where $\hat{s}'_i = \hat{p}_i$ for all $i \in \mathbb{N}_1$. Then, on the left hand side we have:

  $$\hat{t}(\hat{g}' \mid \lambda_{\bar{n},\bar{m}}(g), \lambda_{\bar{n},\bar{m}}(g,a)) = t_{\bar{m}}((\hat{s}_1, \ldots, \hat{s}_k, \hat{s}'_E) \mid (\hat{s}_1, \ldots, \hat{s}_k, \hat{s}_E), a) = t_E(\hat{s}'_E \mid s_E, a)$$

since all of the deterministic agent transitions are 1, so only the transition of the environment is needed to determine the probability. On the right hand side, we note that once again by the deterministic nature of the agent transitions there is precisely one reachable $g' = (s'_1, \ldots, s'_k, \hat{s}'_E) \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{g}')$. Then:

$$\sum_{g' \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{g}')} t_{\bar{n}}(g' \mid g, a) = t_{\bar{n}}((s'_1, \ldots, s'_k, \hat{s}'_E) \mid (s_1, \ldots, s_k, \hat{s}_E), a) = t_E(\hat{s}'_E \mid s_E, a)$$

with the second equality following once again from the deterministic agent transitions being 1. Thus, the result holds.

- $a = (a', (i, j)) \in A_i \times Agt_{\bar{m}}$. We consider only the case when $X = X'$, $\hat{s}_E = \hat{s}'_E$ and $\hat{s}_{l,m} = \hat{s}'_{l,m}$ for all $(l, m) \neq (i, j)$. If any of those conditions are violated, it follows easily from the definitions that both sides are 0. In the remaining case note that:

  $$\hat{t}(\hat{g}' \mid \lambda_{\bar{n},\bar{m}}(g), \lambda_{\bar{n},\bar{m}}(g, a)) = t_i(\hat{s}'_{i,j} \mid \hat{s}_{i,j}, a) = t_i(s'_{i,j} \mid s_{i,j}, a)$$

  by definition. On the right hand side, note that the only non-zero term of the sum is when $g' = (s'_1, \ldots, s'_k, \hat{s}'_E)$ with $s'_{l,m} = s_{l,m}$ for all $(l, m) \neq (i, j)$ and $s'_{i,j} = \hat{s}'_{i,j}$. Thus:

  $$\sum_{g' \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{g}')} t_{\bar{n}}(g' \mid g, a) = t_{\bar{n}}((s'_1, \ldots, s'_k, s'_E) \mid (s_1, \ldots, s_k, s_E), a) = t_i(s'_{i,j} \mid s_{i,j}, a)$$

- $a = (a', (i, j)) \in AE_i \times Agt_{\bar{m}}$. This case is identical to the one above, except that we needn't have $\hat{s}_E = \hat{s}'_E$ and thus the final result is multiplied by $t_E(\hat{s}'_E \mid s_E, a)$ to account for the environment's transition.

- $a = (a', (i, j)) \in A_i \times (Agt_{\bar{n}} \setminus Agt_{\bar{m}})$. We consider only the case when $\hat{s}_E = \hat{s}'_E$ and $\hat{s}_l = \hat{s}'_l$ for all l. If either of those conditions are violated, it follows easily from the definitions that both sides are 0. We further subdivide this case into two:

  - $\{j' \in \{\bar{m}_i + 1, \ldots, \bar{n}_i\} \mid s_{i,j} = s_{i,j'}\} = 1$. Then, $\lambda(g, a) = (a', s_{i,j}, \downarrow)$. Here, there are two non-zero cases. If $X' = X \setminus \{s_{i,j}\}$ then on the LHS we have:

    $$\hat{t}(\hat{g}' \mid \lambda_{\bar{n},\bar{m}}(g), (a', s_{i,j}, \downarrow)) = \sum_{l' \in X \cap S_i} t_i(l' \mid s_{i,j}, a')$$

by definition. But note this is equal to the RHS since the only $g' = (s'_1, \ldots, s'_k, s_E) \in \lambda^{-1}(\hat{g}')$ for which $t_{\bar{n}}(g' \mid g, a)$ is non-zero are those where $s'_o = s_o$ for all $o \neq (i,j)$, and in order to be in the pre-image, it must be the case that $s_{i,j} \in X \cap S_i$.

The other non-zero case is when $X' = (X \setminus \{s_{i,j}\}) \cup \{l'\}$ for some $l' \in S_i \setminus X$. Then, on the LHS we have:

$$\hat{t}(\hat{g}' \mid \lambda_{\bar{n},\bar{m}}(g), (a', s_{i,j}, \downarrow)) = t_i(l' \mid s_{i,j}, a')$$

by definition. But this is equal to the RHS, since the only $g' = (s'_1, \ldots, s'_k, s_E) \in \lambda^{-1}(\hat{g}')$ for which $t_{\bar{n}}(g' \mid g, a)$ is non-zero is the one with $s'_o = s_o$ for all $o \neq (i,j)$ and $s'_{i,j} = l'$.

– $\{j' \in \{\bar{m}_i + 1, \ldots, \bar{n}_i\} \mid s_{i,j} = s_{i,j'}\} > 1$. Then, $\lambda(g,a) = (a', s_{i,j}, \uparrow)$, and once again there are two non-zero cases which are similar to the ones for the case above.

- $a = (a', (i,j)) \in AE_i \times (Agt_{\bar{n}} \setminus Agt_{\bar{m}})$. This case is identical to the one above, except that we needn't have $\hat{s}_E = \hat{s}'_E$ and thus the final result is multiplied by $t_E(\hat{s}'_E \mid s_E, a)$ to account for the environment's transition.

This covers all types of action and completes the proof. □

Having defined the notion of abstraction on states and actions (and proved a desirable property of this) we now go on to extend these definitions to cover paths in a similar way to what we did in the previous chapter – mapping every state and action in the path to the corresponding abstract state.

Definition 5.4 (Path Abstraction)**.** *Let $\bar{n}, \bar{m} \in \mathbb{N}_1^k$ with $\bar{n}_i > \bar{m}_i$ for all i. Then, the abstraction map on paths $\lambda_{\bar{n},\bar{m}} : IPath_{S(\bar{n})} \to IPath_{\hat{\mathcal{S}}(\bar{m})}$ is given by sending each infinite path $\rho = g_0 a_0 g_1 \ldots$ in $S(\bar{n})$ to the unique infinite path $\hat{\rho} = \hat{g}_0 \hat{a}_0 \hat{g}_1 \ldots$ in $\hat{\mathcal{S}}(\bar{m})$ such that, for all $i \in \mathbb{N}$:*

(i) $\hat{g}_i = \lambda_{\bar{n},\bar{m}}(g_i)$;

(ii) $\hat{a}_i = \lambda_{\bar{n},\bar{m}}(g_i, a_i)$.

Note that this is similar to Definition 4.10, however it differs slightly because in (ii) we need to use the state the action is being performed from to decide whether the abstract action should be a growing or a shrinking one. We restrict this definition to finite paths in the obvious way – we map to an abstract path of equal length such that conditions (i) and (ii) hold for all i up to the length of the path.

Having defined this abstraction function, we now define the equivalent strategy. Given a strategy in a concrete system of size $\bar{n}$, the equivalent strategy defines a strategy in the abstract model of size $\bar{m}$ which achieves "the same behaviour" (precisely what we mean by this is formalised later in Lemma 5.2). This definition will be similar to Definition 4.7, but adapted for APMAS.

Definition 5.5 (Equivalent Strategy)**.** *Let* $\sigma : FPath_{S(\bar{n})} \to Dist(Act_{\bar{n}})$ *be a strategy in* $\mathcal{S}(\bar{n})$*. Then, for any* $\bar{m}$ *with* $\bar{m}_i < \bar{n}_i$ *for all* i*, we define the equivalent strategy* $\hat{\sigma} : FPath_{\hat{\mathcal{S}}(\bar{m})} \to Dist(\hat{Act}_{\bar{m}})$ *by:*

$$\hat{\sigma}(\hat{a} \mid \hat{\rho}) \triangleq \frac{\sum_{\rho \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho})} \sum_{a \in Act_{\bar{n}} : \lambda_{\bar{n},\bar{m}}(last(\rho),a) = \hat{a}} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho) \cdot \sigma(a \mid \rho)}{\sum_{\rho \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho})} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho)}$$

Intuitively, as in the abstract model we do not know precisely which path would have been followed in the concrete model, we have to consider all paths that could have been followed (these are precisely the paths in the preimage of $\lambda_{\bar{n},\bar{m}}$) and weight our probabilistic choice of the next action according to the probability of each path.

Before we proceed any further, we should stop to prove that this definition is indeed a valid strategy. This proof is similar to that for Observation 4.2, but modified for the case of APMAS.

Observation 5.2. $\hat{\sigma} : FPath_{\hat{\mathcal{S}}(\bar{m})} \to Dist(\hat{Act}_{\bar{m}})$ *is a valid strategy.*

Proof. First, suppose that $\hat{\sigma}(\hat{a} \mid \hat{\rho}) > 0$. Then, there is at least one $\rho \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho})$ and one $a \in Act_{\bar{n}}$ such that $\lambda_{\bar{n},\bar{m}}(last(\rho), a) = \hat{a}$ and $\sigma(a \mid \rho) > 0$. Then it must be the case that $a \in P_{\bar{n}}(last(\rho))$ since σ is a valid strategy. It follows that $\hat{a} \in \hat{P}_{\bar{n}}(last(\hat{\rho}))$

It remains to check that for any $\hat{\rho} \in FPath_{\hat{\mathcal{S}}(\bar{m})}$ we have that $\sum_{\hat{a} \in \hat{Act}_{\bar{m}}} \hat{\sigma}(\hat{a} \mid \hat{\rho}) = 1$. For this notice that:

$$\sum_{\rho \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho})} \sum_{\hat{a} \in \hat{Act}_{\bar{m}}} \sum_{a \in Act_n : \lambda_{\bar{n},\bar{m}}(last(\rho),a) = \hat{a}} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho) \cdot \sigma(a \mid \rho)$$

$$= \sum_{\rho \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho})} \sum_{a \in Act_{\bar{n}}} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho) \cdot \sigma(a \mid \rho) = \sum_{\rho \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho})} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho)$$

with the first equality following simply from $\lambda_{\bar{n},\bar{m}}$ being a function and the second from σ being a valid strategy. So $\sum_{\hat{a} \in \hat{Act}_{\bar{m}}} \hat{\sigma}(\hat{a} \mid \hat{\rho}) = 1$, as desired. □

Having proved the validity of our definition, we present a further lemma. This shows the desirable property that when following the equivalent strategy, the probability of a path in the abstract model is exactly equal to the sum of the probabilities of the concrete paths that it could represent.

Lemma 5.2. *Let $\sigma : FPath_{S(\bar{n})} \to Dist(Act_{\bar{n}})$ be a strategy in $\mathcal{S}(\bar{n})$. Then it is the case that, for any path $\hat{\rho} \in FPath_{\hat{\mathcal{S}}(\bar{m})}$:*

$$\mathbf{P}_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}}(\hat{\rho}) = \sum_{\rho \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho})} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho)$$

Proof. We proceed by induction on the length of the path $\hat{\rho}$. For the base case, note the only path of length 1 has just the initial state $\hat{\iota}_{\bar{m}}$. This has probability 1. Further, $\rho = \iota$ is the only element of $\lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho})$ for which $\mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho)$ is non-zero, and this is also 1. So, both sides of the equation are 1, and the statement holds.

Now, suppose the statement holds for a path $\hat{\rho}'$ and consider its extension $\hat{\rho} = \hat{\rho}'\hat{a}\hat{s}$ for

some $\hat{a} \in \hat{Act}_{\bar{m}}$ and $\hat{s} \in \hat{S}_{\bar{m}}$. Now:

$$\begin{aligned}
&\mathbf{P}_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}}(\hat{\rho}) \\
&\quad = \mathbf{P}_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}}(\hat{\rho}') \cdot \frac{\sum_{\rho \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho}')} \sum_{a \in Act_{\bar{n}} : \lambda_{\bar{n},\bar{m}}(last(\rho),a) = \hat{a}} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho) \cdot \sigma(a \mid \rho)}{\sum_{\rho \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho}')} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho)} \cdot \hat{t}_{\bar{m}}(\hat{s} \mid last(\hat{\rho}'), \hat{a}) \\
&\quad = \left(\sum_{\rho \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho}')} \sum_{a \in Act_{\bar{n}} : \lambda_{\bar{n},\bar{m}}(last(\rho),a) = \hat{a}} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho) \cdot \sigma(a \mid \rho) \right) \cdot \hat{t}_{\bar{m}}(\hat{s} \mid last(\hat{\rho}'), \hat{a}) \\
&\quad = \sum_{\rho \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho}')} \sum_{a \in Act_{\bar{n}} : \lambda_{\bar{n},\bar{m}}(last(\rho),a) = \hat{a}} \left(\mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho) \cdot \sigma(a \mid \rho) \cdot \sum_{s \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{s})} t_{\bar{n}}(s \mid last(\rho), a) \right) \\
&\quad = \sum_{\rho \in \lambda^{-1}_{\bar{n},\bar{m}}(\hat{\rho})} \mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\rho)
\end{aligned}$$

with the first equality following from the definitions, the second from the inductive hypothesis, the third from Lemma 5.1 and the final one simply by rearranging. Thus, by induction on the length of the path, the lemma holds for all finite paths. □

Having proved this lemma, we now have enough to prove one of the main results of this chapter. In particular, we will now prove that if an $\bar{m}$-indexed formula of the form $P^{\max}_{\leq x}[\psi]$ holds in the abstract model of size $\bar{m}$ then it will hold in any larger concrete system. This will enable us to partially solve the PMCP from Definition 5.1.

Theorem 5.1. *Suppose $\hat{\mathcal{S}}(\bar{m}) \models P^{max}_{\leq x}[\psi]$ for some $\bar{m}$-indexed formula ψ. Then, $\mathcal{S}(\bar{n}) \models P^{max}_{\leq x}[\psi]$ for all $\bar{n} \in \mathbb{N}^k_1$ with $\bar{n}_i > \bar{m}_i$ for all i.*

Proof. We prove the contrapositive of this statement. Suppose we have $\mathcal{S}(\bar{n}) \models P^{\max}_{>x}[\psi]$ for some $\bar{n} \in \mathbb{N}^k_1$ with $\bar{n}_i > \bar{m}_i$ for all i. Then, by definition, for some strategy σ in $\mathcal{S}(\bar{n})$ we have:

$$\mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\{\rho \in IPath_{\mathcal{S}(\bar{n})_\sigma} : \rho \models \psi\}) > x$$

Now, let $\hat{\sigma}$ be the equivalent strategy in $\hat{\mathcal{S}}(\bar{m})$ as given by Definition 5.5. But now every path in the concrete model has a corresponding path in the abstract model where, by definition,

the same atomic propositions hold at each step (and thus the same LTL formulas hold along the path). So:

$$\mathbf{P}_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}}(\{\hat{\rho} \in IPath_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}} : \hat{\rho} \models \psi\}) > x$$

since by Lemma 5.2 the probability of a path in the abstract model is at least as much as that of the corresponding path in the concrete model.

Thus, by considering $\hat{\sigma}$ as our strategy, we see that $\hat{\mathcal{S}}(\bar{m}) \models P^{\max}_{>x}[\psi]$ and our result holds. □

Note the above theorem also gives a method to check properties of the form $P^{\min}_{>x}[\psi]$ since this is equivalent to checking $P^{\max}_{\leq 1-x}[\neg\psi]$. Further, if we can verify $P^{\max}_{\leq x}[\psi]$ then certainly it cannot be the case that $P^{\max}_{>x}[\psi]$. This enables us to falsify properties of the form $P^{\max}_{>x}[\psi]$ (and thus also properties of the form $P^{\min}_{\leq x}[\psi]$, which are equivalent to $P^{\max}_{>1-x}[\neg\psi]$).

Theorem 5.2. *Suppose $\hat{\mathcal{S}}(\bar{m}) \models P^{max}_{<x}[\psi]$ for some $\bar{m}$-indexed formula ψ. Then, $\mathcal{S}(\bar{n}) \models P^{max}_{<x}[\psi]$ for all $\bar{n} \in \mathbb{N}^k_1$ with $\bar{n}_i > \bar{m}_i$ for all i.*

Proof. This is identical to the proof for Theorem 5.1 with strict and non-strict inequalities swapped. □

As before, this theorem can also be used to verify properties of the form $P^{\min}_{\geq x}[\psi]$, and to falsify properties of the form $P^{\max}_{\geq x}[\psi]$ or $P^{\min}_{<x}[\psi]$.

5.3 Verification Procedure

All the above combine to give our partial decision procedure for solving the PMCP, which can be seen in Algorithm 5.1. We now prove its correctness.

Corollary 5.1. *Let $\mathcal{S}$ be an APMAS, and ϕ an $\bar{m}$-indexed PLTL formula. Then, if* `PMCP-Unbounded`$(\mathcal{S}, \phi)$ *= true, it is always the case that $\mathcal{S} \models \phi$. Further, if* `PMCP-Unbounded`$(\mathcal{S}, \phi) =$ *false, then it is the case that $\mathcal{S} \not\models \phi$.*

Algorithm 5.1 Decision procedure for the PMCP of APMAS against PLTL

Input: APMAS $\mathcal{S}$, PLTL formula ϕ

Output: Boolean, or `FAIL`

function `PMCP-UNBOUNDED`($\mathcal{S}$, ϕ)
  $\bar{m} \leftarrow$ `ComputeIndex`(ϕ)
  switch ϕ **do**
    case $\phi = P^{\max}_{\leq x}[\psi]$ or $\phi = P^{\max}_{<x}[\psi]$
      if $\hat{\mathcal{S}}(\bar{m}) \not\models \phi$ **then**
        return `FAIL`
      end if
      return *true*
    case $\phi = P^{\max}_{>x}[\psi]$
      if `Check`($\mathcal{S}$, $P^{\max}_{\leq x}[\psi]$) = *true* **then**
        return *false*
      end if
      return `FAIL`
    case $\phi = P^{\max}_{\geq x}[\psi]$
      if `Check`($\mathcal{S}$, $P^{\max}_{<x}[\psi]$) = *true* **then**
        return *false*
      end if
      return `FAIL`
    case $\phi = P^{\min}_{>x}[\psi]$
      return `Check`($\mathcal{S}$, $P^{\max}_{\leq 1-x}[\neg\psi]$)
    case $\phi = P^{\min}_{\geq x}[\psi]$
      return `Check`($\mathcal{S}$, $P^{\max}_{<1-x}[\neg\psi]$)
    case $\phi = P^{\min}_{<x}[\psi]$
      return `Check`($\mathcal{S}$, $P^{\max}_{\geq 1-x}[\neg\psi]$)
    case $\phi = P^{\min}_{\leq x}[\psi]$
      return `Check`($\mathcal{S}$, $P^{\max}_{>1-x}[\neg\psi]$)
end function

Proof. If we return *true* on line 8, then by either Theorem 5.1 or Theorem 5.2 (depending on the form of ϕ), we know that $\mathcal{S}(\bar{n}) \models \phi$ for all $\bar{n} \in (\mathbb{N}_1)^k$ with $\bar{n}_i > \bar{m}_i$ for all i, showing that $\mathcal{S} \models \phi$.

If we return on line 11, then $\phi = P^{max}_{>x}[\psi]$ and (by the reasoning in the previous parts of the proof) we have proved that we must have $\mathcal{S} \models P^{max}_{\leq x}[\psi]$. Thus, it must be the case that $\mathcal{S} \not\models \phi$. The same argument proves the return value on line 16.

For the return value on line 20, note that checking $P^{min}_{>x}[\psi]$ is identical to checking $P^{max}_{\leq 1-x}[\neg\psi]$ since the strategy that minimises ψ is just the one that maximises $\neg\psi$, and if this achieves a probability $\leq 1 - x$ of satisfying $\neg\psi$, it will certainly achieve a probability $> x$ of satisfying ψ. A similar argument shows the validity of the return values on lines 22, 24 and 26. □

Thus, we have proved that our procedure's result (when it returns one) is correct. Further, the procedure always terminates. This is clear since when it makes a recursive call this call is always to one of the cases above it, and further there are no loops. However, the procedure is not complete since in some cases it will not return a result but will return `FAIL` instead. This is inevitable, since the PMCP decision problem it is solving (Definition 5.1) is a more general version of one that is already known to be undecidable [Apt and Kozen, 1986] and thus is also undecidable. As we did for the procedure in the previous section, we now make an observation on our procedures run-time.

Observation 5.3. *The run-time of* `PMCP-Unbounded`$(\mathcal{S}, \phi)$ *is exponential in the number of states in agent templates, polynomial in the number of environment states, and doubly exponential in the size of the formula.*

Proof. The proof is identical to that of Observation 4.4. □

5.4 Summary

In this chapter, we have developed a technique to solve the PMCP of APMAS against PLTL specifications (Definition 5.1). To achieve this, we first defined an abstract model for APMAS (Definition 5.2). We then proved that to check properties of larger systems, it is sufficient to verify them in the abstract model (Theorems 5.1 and 5.2). This result allowed us to develop a verification procedure (Algorithm 5.1) which we proved to be sound (Corollary 5.1).

Chapter Six

Extensions

In this chapter, we will introduce two extensions to our semantics that will allow us to model richer scenarios. In particular, in the first section we will consider strategic properties that allow us to express goals that a coalition of agents may achieve. Subsequently, in the second section we will introduce a framework to inject faults in the behaviour of the agents, thus allowing us to model situations where agents may malfunction.

The first part of this chapter is based on research that was first presented in our 2020 AAMAS paper [Lomuscio and Pirovano, 2020a], while the second part was first presented in our 2020 IJCAI paper [Lomuscio and Pirovano, 2020b].

6.1 Strategic Specifications

In this section we introduce a parameterised model checking problem for unbounded probabilistic systems against strategic properties. We then develop a partial decision procedure for this problem which relies on computing upper and lower bounds for what probabilities coalitions of agents can achieve of satisfying a certain path formula. Some of these bounds will be computed using the abstract model we introduced in Section 4.2.

6.1.1 Parameterised Model Checking Problem

As we have done in previous chapters, we now introduce the parameterised model checking problem that we will consider for the rest of this section. This problem will be similar to the one in Definition 4.1 but using the richer specification logic of P[ATL*].

Definition 6.1 (PMCP of P[ATL*] on SPMAS)**.** *Given an SPMAS $\mathcal{S}$ and an $\bar{m}$-indexed P[ATL*] formula ϕ, the parameterised model checking problem (PMCP) involves establishing whether it is the case that $\mathcal{S}(\bar{n}) \models \phi$ for all $\bar{n}$ with $\bar{n}_i > \bar{m}_i$ for all i. We write $\mathcal{S} \models \phi$ if this is the case.*

Once again, note that this decision problem is an extension (to incorporate probabilities and strategies) of one that is already know to be undecidable [Apt and Kozen, 1986]. Thus, it is also undecidable. Nonetheless, we devote the rest of this section to developing a partial decision procedure for it.

6.1.2 Bounding the Maximal Probability

In this subsection, we aim to compute lower and upper bounds on the maximal probability with which a coalition of agents can enforce a path formula. Before doing so, we need to define what we mean by this.

Definition 6.2 (Maximal Probability)**.** *Let $\mathcal{S}$ be an SPMAS, A a coalition of agents and ψ a path formula. Then we use $\langle\langle A\rangle\rangle P_{\bar{n},max=?}[\psi]$ to denote the maximal value of $r \in [0,1]$ for which it is the case that $\mathcal{S}(\bar{n}) \models \langle\langle A\rangle\rangle P_{\geq r}[\psi]$.*

Intuitively, in the above definition $\langle\langle A\rangle\rangle P_{\bar{n},max=?}[\psi]$ is the maximum probability with which the agents A can ensure ψ is achieved in a system of size $\bar{n}$. Note that since the system of size $\bar{n}$ is finite, this is well-defined and there is a strategy that achieves it.

Observe that if we can compute the minimum and maximum values for $\langle\langle A\rangle\rangle P_{\bar{n},max=?}[\psi]$ as we range over $\bar{n}$, we can obtain a decision procedure for the PMCP against formulas of the

form $\langle\langle A\rangle\rangle P_{\geq r}[\psi]$ or $\langle\langle A\rangle\rangle P_{>r}[\psi]$. We will explore how to do this in this subsection, before providing symmetric results for minimum values in the next subsection.

The following result gives the maximum value as it shows that $\langle\langle A\rangle\rangle P_{\bar{n},max=?}[\psi]$ is non-increasing.

Lemma 6.1. *Let $\mathcal{S}$ be an SPMAS. Then, for any set of agents A and path formula ψ it is the case that:*

$$\langle\langle A\rangle\rangle P_{\bar{m},max=?}[\psi] \geq \langle\langle A\rangle\rangle P_{\bar{n},max=?}[\psi]$$

where $\bar{n}_i \geq \bar{m}_i$ for all $i \in \dot{k}$ and $\bar{m}$ is at least the index of the formula.

Proof. Consider the strategy σ'_A for the agents in A that achieves the maximal probability $r = \langle\langle A\rangle\rangle P_{\bar{n},max=?}$ in the larger system of size $\bar{n}$. Denote by σ_A the stategy for the agents in A in the system of size $\bar{m}$ that behaves the same way. We claim that for any strategy σ_{A^c} for the remaining agents in the system of size $\bar{m}$ it is the case that

$$\mathbf{P}_{\mathcal{S}(\bar{m})_\sigma}(\{\omega \in \mathrm{IPath}_{\mathcal{S}(\bar{m})_\sigma}(\iota) : \omega \models \psi\}) \geq r$$

where σ is the complete strategy given by σ_A and σ_{A^c}. Suppose for a contradiction that this were not the case for some strategy σ_{A^c}. Now let σ'_{A^c} denote the stategy obtained by extending σ_{A^c} to the larger system of size $\bar{n}$ by having all the extra agents always perform the null action ε. Now, note that

$$\mathbf{P}_{\mathcal{S}(\bar{n})_{\sigma'}}(\{\omega \in \mathrm{IPath}_{\mathcal{S}(\bar{n})_{\sigma'}}(\iota) : \omega \models \psi\}) < r$$

where σ' is the complete strategy given by σ'_A and σ'_{A^c}. This gives our desired contradiction, completing the proof. □

This lemma is a probabilistic equivalent of the intuitive property that in a non-probabilistic system with null actions, adding an agent that is not part of the coalition trying to achieve a formula will not make it satisfied if it was not already. In particular, the lemma shows that if the agents A have a strategy to achieve a certain probability of success

in the system of size $\bar{n}$, then they will also have a strategy to achieve at least this probability in the smaller system of size $\bar{m}$.

It follows from the lemma that to compute the maximum value of $\langle\langle A\rangle\rangle P_{\bar{n},max=?}[\psi]$ as we vary $\bar{n}$ for an $\bar{m}$-indexed formula, it suffices to compute the value of $\langle\langle A\rangle\rangle P_{\bar{m},max=?}[\psi]$. Note that it is immediate that this maximum is attained, since the system of size $\bar{m}$ achieves it. We now define a concept analogous to Definition 6.2 for the abstract system.

Definition 6.3 (Maximal Abstract Probability)**.** *Let $\mathcal{S}$ be an SPMAS, A a coalition of agents and ψ a path formula. Then we use $\langle\langle A\rangle\rangle \hat{P}_{\bar{n},max=?}[\psi]$ to denote the maximal value of $r \in [0,1]$ for which it is the case that $\hat{\mathcal{S}}(\bar{n}) \models \langle\langle A\rangle\rangle P_{\geq r}[\psi]$ where $\hat{\mathcal{S}}$ is the abstract model defined in Definition 4.2.*

Intuitively, this defines the maximum probability with which agents in A can ensure the property ψ in the abstract system of size $\bar{n}$. This brings us to the other main result of this section, which gives a lower bound on the values of $\langle\langle A\rangle\rangle P_{\bar{n},max=?}[\psi]$.

Lemma 6.2. *Let $\mathcal{S}$ be an SPMAS. Then, for any set of agents A and path formula ψ it is the case that:*

$$\langle\langle A\rangle\rangle P_{\bar{n},max=?}[\psi] \geq \langle\langle A\rangle\rangle \hat{P}_{\bar{m},max=?}[\psi]$$

where $\bar{m}$ is the index of the formula and $\bar{n}_i > \bar{m}_i$ for all $i \in \dot{k}$.

Proof. Let $\hat{\sigma}_A$ denote the strategy for the agents A in $\hat{\mathcal{S}}(\bar{m})$ that achieves the maximum probability $r = \langle\langle A\rangle\rangle \hat{P}_{\bar{m},max=?}[\psi]$. Consider the strategy σ_A in $\mathcal{S}(\bar{n})$ which behaves in the same way. We claim that for any strategy σ_{A^c} for the remaining agents in $\mathcal{S}(\bar{n})$, it is the case that:

$$\mathbf{P}_{\mathcal{S}(\bar{n})_\sigma}(\{\omega \in \mathrm{IPath}_{\mathcal{S}(\bar{n})_\sigma}(\iota) : \omega \models \psi\}) \geq r$$

where σ is the complete strategy given by σ_A and σ_{A^c}. Suppose for a contradiction that this is not the case for some σ_{A^c}. Now, define $\hat{\sigma}_{A^c}$ as the equivalent strategy according to

Definition 4.7. Then, it follows from Lemma 4.2 that:

$$\mathbf{P}_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}}(\{\hat{\rho} \in IPath_{\hat{\mathcal{S}}(\bar{m})_{\hat{\sigma}}} : \hat{\rho} \models \psi\}) < r$$

where $\hat{\sigma}$ is the complete strategy given by $\hat{\sigma}_A$ and $\hat{\sigma}_{A^c}$. This gives a contradiction as desired.

□

Intuitively, the abstract model represents a more powerful opponent for the agents in A than any concrete system, since within the second component it captures the possible behaviours of an arbitrarily large number of agents. Thus, if a coalition of agents has a strategy to achieve a certain probability in the abstract system of size $\bar{m}$, they must also have a strategy to achieve at least this probability in the concrete system of size $\bar{n}$.

6.1.3 Bounding the Minimal Probability

Having defined in the previous subsection some bounds for the maximal probabilities that coalitions of agents can enforce, we briefly outline the symmetric results for minimal values. Firstly, we repeat Definition 6.2 for minimum values.

Definition 6.4 (Minimum Probability)**.** *Let $\mathcal{S}$ be an SPMAS, A a coalition of agents and ψ a path formula. Then we use $\langle\langle A\rangle\rangle P_{\bar{n},min=?}[\psi]$ to denote the minimal value of $r \in [0,1]$ for which it is the case that $\mathcal{S}(\bar{n}) \models \langle\langle A\rangle\rangle P_{\leq r}[\psi]$.*

As in Definition 6.3, we write $\langle\langle A\rangle\rangle \hat{P}_{\bar{n},min=?}[\psi]$ for the variation of the above definition to the abstract model $\hat{\mathcal{S}}$. We now give equivalents of Lemmas 6.1 and 6.2 for minimum values.

Lemma 6.3. *Let $\mathcal{S}$ be an SPMAS. Then, for any set of agents A and path formula ψ it is the case that:*

$$\langle\langle A\rangle\rangle P_{\bar{m},min=?}[\psi] \leq \langle\langle A\rangle\rangle P_{\bar{n},min=?}[\psi]$$

where $\bar{n}_i \geq \bar{m}_i$ for all $i \in \dot{k}$ and $\bar{m}$ is at least the index of the formula.

Lemma 6.4. *Let $\mathcal{S}$ be an SPMAS. Then, for any set of agents A and path formula ψ it is the case that:*

$$\langle\langle A\rangle\rangle P_{\bar{n},min=?}[\psi] \leq \langle\langle A\rangle\rangle \hat{P}_{\bar{m},min=?}[\psi]$$

where $\bar{m}$ is the index of the formula and $\bar{n}_i > \bar{m}_i$ for all $i \in \dot{k}$.

We omit the proofs for these two lemmas, which are similar to those for Lemmas 6.1 and 6.2 but with inequalities reversed.

6.1.4 Verification Procedure

Having given versions of our lemmas for the minimum cases, we can now combine all our results into a decision procedure for strategic properties, which is shown in Algorithm 6.1. We now prove its correctness.

Theorem 6.1. *For any SPMAS $\mathcal{S}$ and P[ATL*] formulas ϕ for which* `PMCP-Strategy` *returns a value, it is the case that $\mathcal{S} \models \phi$ iff* `PMCP-Strategy`$(\mathcal{S}, \phi) = true$.

Proof. The correctness of the return values on lines 7 and 11 follow from Lemma 6.1 and Lemma 6.2, respectively. For the return values on lines 17 and 21, the correctness follows from Lemma 6.3 and Lemma 6.4, respectively. □

Algorithm 6.1 Decision procedure for the PMCP of SPMAS against P[ATL*]

Input: SPMAS $\mathcal{S}$, P[ATL*] formula ϕ

Output: Boolean, or `FAIL`

function PMCP-STRATEGY($\mathcal{S}$, ϕ)
  $\bar{m} \leftarrow$ `ComputeIndex`(ϕ)
  switch ϕ **do**
    case $\phi = \langle\langle A\rangle\rangle P_{\bowtie r}[\psi]$ for $\bowtie \in \{\geq, >\}$
      $upper \leftarrow \langle\langle A\rangle\rangle P_{\bar{m}+\bar{1},max=?}[\psi]$
      if $r \bowtie upper$ **then**
        return *false*
      end if
      $lower \leftarrow \langle\langle A\rangle\rangle \hat{P}_{\bar{m},max=?}[\psi]$
      if $lower \bowtie r$ **then**
        return *true*
      end if
      return `FAIL`
    case $\phi = \langle\langle A\rangle\rangle P_{\bowtie r}[\psi]$ for $\bowtie \in \{\leq, <\}$
      $lower \leftarrow \langle\langle A\rangle\rangle P_{\bar{m}+\bar{1},min=?}[\psi]$
      if $r \bowtie lower$ **then**
        return *true*
      end if
      $upper \leftarrow \langle\langle A\rangle\rangle \hat{P}_{\bar{m},min=?}[\psi]$
      if $upper \bowtie r$ **then**
        return *false*
      end if
      return `FAIL`
end function

6.2 Faulty Systems

In this section, we introduce a method for reasoning about how faults that may be exhibited by agents can affect the satisfaction of a specification. In order to do so, we first extend a notion of faults from safety analysis to our semantics. After doing this, we define how to obtain a faulty system from a non-faulty one and a description of the faults that may occur. We then extend this definition to a probabilistically faulty system, in which rather than all agents exhibiting faults we only wish for each agent to exhibit faults with a certain probability.

6.2.1 Fault Injection

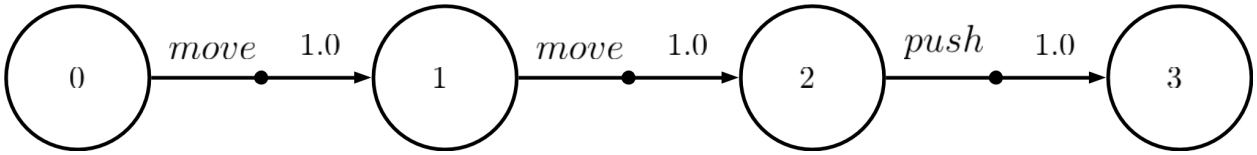
In order to model faults, we assume that the local states S_i of each agent type i are defined by a set of integer, Boolean, and enumerate (over a domain Ω_i) variables $VAR_i = BVar_i \cup IVar_i \cup EVar_i$. More formally, we write

$$S_i = (bool : BVar_i \to \{\bot, \top\}) \times (int : IVar_i \to \mathbb{Z}) \times (enum : EVar_i \to \Omega_i)$$

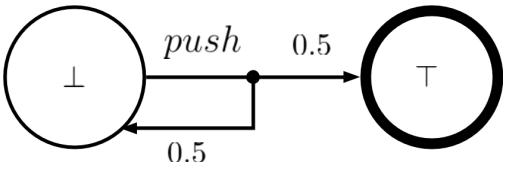
With a slight abuse of notation, we will simply write the values of the variables directly when this is clear from context.

Before proceeding with our definition of how faults are injected, we introduce an example that we will use throughout this section to illustrate the definitions being presented. This is inspired by the autonomous robot scenario [Fagin et al., 1995] but adapted to include probabilities.

Example 6.1 (Robots on track)**.** *A group of robots begin at the left end of a track of length* $k+1$*. A robot at a position* $x < k$ *can (independently of other robots) choose to move right one unit, which it will do with probability* 1*. At position* k *there is a button which the robots can press. The button press is successful with probability* 0.5*. Regardless of whether the press is successful or not, the robot that pressed it is moved to the right. Once the button has been*



(a) An example agent template.



(b) An example environment.

Figure 6.1 The APMAS for our version of the autonomous robots scenario with $k = 2$. The *move* action is an asynchronous one, whilst the *push* action is an agent-environment one. The atomic proposition *buttonPressed* holds when the environment is in the bold state.

successfully pressed, no other robot can press it. Note the robots can never move left, so each robot only gets one chance to press the button. The APMAS that models this scenario (for a track of length 3) can be seen in Figure 6.1.

We denote by $\mathcal{F}_i$ the set of all possible faults for agents of type i. Various studies have been conducted in safety analysis to identify the faults normally of interest [Bozzano and Villafiorita, 2007]. We here consider the most widely used faults by assuming that $\mathcal{F}_i$ contains:

- For every $x \in BVar_i$, a fault $invert(x)$ that inverts the value of x and a fault $setB(x, k)$ (where $k \in \{\bot, \top\}$) that sets the value of x to k.
- For every $y \in IVar_i$, a fault $up(y)$ that increments the value of y, a fault $down(y)$ that decrements the value of y, and a fault $setI(y, k)$ (where $k \in \mathbb{Z}$) that sets the value of y to k.
- For every $z \in EVar_i$, a fault $setE(z, v)$ (where $v \in \Omega_i$) that sets the value of z to v.

More formally, $\mathcal{F}_i$ is given by:

$$\begin{aligned}\mathcal{F}_i \triangleq &\{invert(x), setB(x,k) : x \in BVar_i, k \in \{\top, \bot\}\} \\ &\cup \{up(y), down(y), setI(y,k) : y \in IVar_i, k \in \mathbb{Z}\} \\ &\cup \{setE(z,v) : z \in EVar_i, v \in \Omega_i\}\end{aligned}$$

Given a fault $f \in \mathcal{F}_i$ and a state $s \in S_i$, we will denote by $(s)_f$ the result of applying f to s. Formally:

$$((bool, int, enum))_f = \begin{cases} (bool_{x \mapsto \neg bool(x)}, int, enum) & \text{if } f = invert(x) \\ (bool_{x \mapsto k}, int, enum) & \text{if } f = setB(x,k) \\ (bool, int_{y \mapsto int(y)+1}, enum) & \text{if } f = up(y) \\ (bool, int_{y \mapsto int(y)-1}, enum) & \text{if } f = down(y) \\ (bool, int_{y \mapsto k}, enum) & \text{if } f = setI(y,k) \\ (bool, int, enum_{z \mapsto v}) & \text{if } f = setE(z,v) \end{cases}$$

where $g_{x \mapsto y}$ denotes the function obtained by replacing the value of g at x with y.

6.2.2 Fully Faulty Systems

In this subsection, we will build on the above definitions of faults to explore how a faulty system can be constructed from a non-faulty one and a description of the faults. This notion of fault injection is similar to the one previously considered in the literature [Bozzano and Villafiorita, 2007; Ezekiel and Lomuscio, 2017]. However, we note the important difference that we here consider unbounded stochastic systems, with stochastic faults. To the best of our knowledge, this has not been studied before as without the foundation of the work carried out in the previous chapters of this thesis it is a difficult problem to tackle.

Before defining faulty systems, we introduce the notion of fault profiles, which define the probability of each fault occurring when performing a certain action from a certain state.

These fault profiles will complete the necessary information to build faulty systems from non-faulty ones.

Definition 6.5 (Fault Profile)**.** *A fault profile is a function* $\chi_i : S_i \times Act_i \to Dist(\mathcal{F}_i \cup \{\checkmark\})$*. The expression* $\chi_{ii}(s, a)$ *gives a probability distribution on what fault will occur when action* a *is performed from state* s*, with the* $\checkmark$ *being used to denote no fault occurring.*

Note that the above restricts our systems to having at most one fault at each time-step. This restriction could be lifted by considering sets of faults instead of individual faults in the fault profile. For ease of presentation we do not pursue this here.

We now go on to define how an agent template can be modified to exhibit the faults given by a fault profile.

Definition 6.6 (Faulty Agent)**.** *Given an agent template* $T_i = \langle S_i, \iota_i, Act_i, P_i, t_i\rangle$ *and a fault profile* $\chi_i : S_i \times Act_i \to Dist(\mathcal{F}_i \cup \{\checkmark\})$*, we define the faulty agent template* $T_i^{\chi_i} = \langle S_i^{\chi_i}, \iota_i^{\chi_i}, Act_i, P_i^{\chi_i}, t_i^{\chi_i}\rangle$ *as follows:*

- $S_i^{\chi_i} = S_i \times \{\bot, \top\} \times \{\bot, \top\}$*, where the first Boolean variable encodes whether the agent has ever exhibited a fault and the second whether the agent exhibited a fault in the previous transition.*
- $\iota_i^{\chi_i} = (\iota_i, \bot, \bot)$ *is a new initial state.*
- $P_i^{\chi_i} : S_i^{\chi_i} \to \mathcal{P}(Act_i)$ *is given by* $P_i^{\chi_i}((s, f, i)) \triangleq P_i(s)$*.*
- $t_i^{\chi_i} : S_i^{\chi_i} \times Act_i \to Dist(S^{\chi_i})$ *is defined by:*

$$t_i^{\chi_i}((s', f', e') \mid (s, f, e), a) \triangleq \begin{cases} t_i(s' \mid s, a)\chi_i(\checkmark \mid s, a) & \text{if } f = f' \text{ and } e' = \bot \\ \sum_{x \in \mathcal{F}_i, s'' \in S_i : (s'')_x = s'} t_i(s'' \mid s, a)\chi_i(x \mid s, a) & \text{if } f' = e' = \top \\ 0 & \text{otherwise} \end{cases}$$

The definition above includes both non-faulty transitions (corresponding to $e' = \bot$) in which the value of f is not affected, and faulty transitions ($e' = \top$) in which f is set to $\top$. Note that transitions where $f = \top$ and $f' = \bot$ are not allowed; so once an agent has exhibited faulty behaviour, it is labelled as faulty for the rest of the run of the system. However, faulty agents may still carry out correct transitions. Further, note that faults are never injected into our two new variables that track faults.

Example 6.2. *Consider again the track scenario from Example 6.1. Suppose we wish to model that with probability* 0.2 *when a robot moves it may malfunction and move two units instead of one.*

To model this in our framework, let x be the variable representing the location of the robot in its state. Then, we could consider a fault profile $\chi_i : S_i \times Act_i \to Dist(\mathcal{F}_i \cup \{\checkmark\})$ of:

$$\chi_i(f \mid s, a) \triangleq \begin{cases} 0.2 & \text{if } a = move \text{ and } f = up(x) \\ 0.8 & \text{if } a = move \text{ and } f = \checkmark \\ 1 & \text{if } a \neq move \text{ and } f = \checkmark \\ 0 & \text{otherwise} \end{cases}$$

The faulty agent resulting from applying our construction (with a track of length 3) can be seen in Figure 6.2.

Before proceeding, we need to show that our definition of faulty agents is valid. In particular, we should check that $t_i^{\chi_i}$ defines a valid probability distribution.

Observation 6.1. *Let $\chi_i : S_i \times Act_i \to Dist(\mathcal{F}_i \cup \{\checkmark\})$ be a fault profile and $T_i = \langle S_i, \iota_i, Act_i, P_i, t_i \rangle$ a non-faulty agent. Then, it is the case that for any $l = (s, f, e) \in S_i^{\chi_i}$ and $a \in Act_i$ we have $\sum_{l' \in S_i^{\chi_i}} t_i^{\chi_i}(l' \mid l, a) = 1$.*

Proof. Note that we only have two non-zero cases in our transition function (a faulty tran-

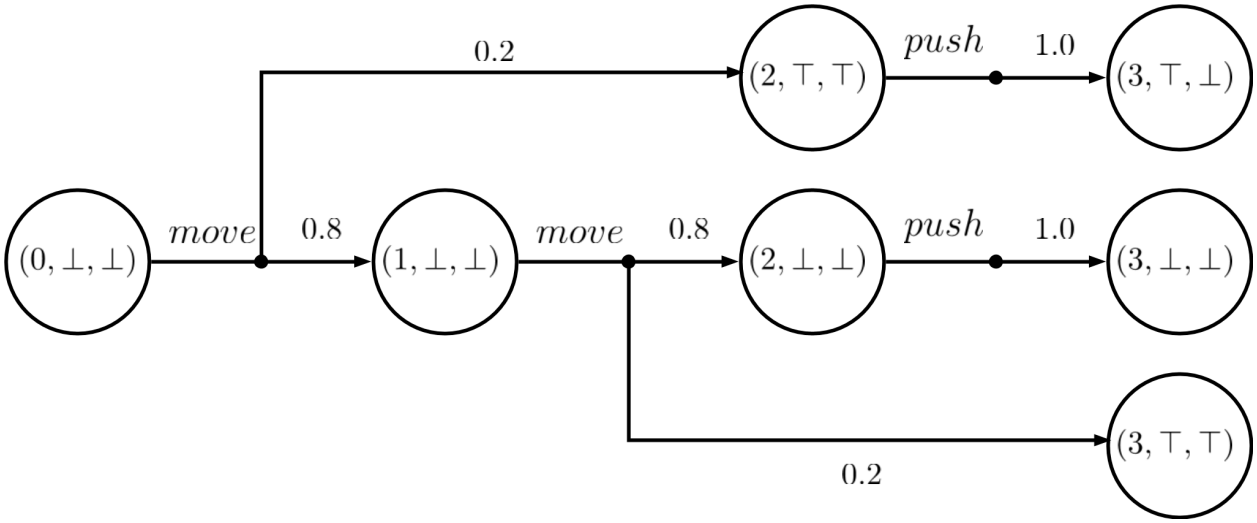


Figure 6.2 An example faulty agent template resulting from adding a fault (corresponding to the *move* action moving the agent two units instead of one with probability 0.2) to the agent in Figure 6.1.

sition and a non-faulty one). So, the sum on the LHS can be split into:

$$\sum_{l' \in S^{\chi_i} : l' = (s', \top, \top)} t_i^{\chi_i}(l' \mid l, a) + \sum_{l' \in S^{\chi_i} : l' = (s', f, \bot)} t_i^{\chi_i}(l' \mid l, a)$$

Then, we can apply the definition of the transition function and rearrange to get:

$$\sum_{x \in \mathcal{F}_i} \chi_i(x \mid s, a) \sum_{l' \in S_i^{\chi_i} : l' = (s', \top, \top)} \sum_{s'' \in S_i : (s'')_x = s'} t(s'' \mid s, a)$$
$$+ \chi_i(\checkmark \mid s, a) \sum_{l' \in S_i^{\chi_i} : l' = (s', f, \bot)} t_i(s' \mid s, a)$$

We can simplify some of the subscripts by noting that the second two components of the tuple are not needed to get:

$$\sum_{x \in \mathcal{F}_i} \chi_i(x \mid s, a) \sum_{s' \in S_i} \sum_{s'' \in S_i : (s'')_x = s'} t_i(s'' \mid s, a) + \chi_i(\checkmark \mid s, a) \sum_{s' \in S_i} t_i(s' \mid s, a)$$

Now, note that since s' ranges over all possible states we have that:

$$\sum_{s' \in S_i} \sum_{s'' \in S_i : (s'')_x = s'} t_i(s'' \mid s, a) = \sum_{s'' \in S_i} t_i(s'' \mid s, a) = 1$$

with the second equality following from t being a valid transition function. So, the LHS is equal to:

$$\sum_{x \in \mathcal{F}_i} \chi_i(x \mid s, a) + \chi_i(\checkmark \mid s, a) = \sum_{x \in \mathcal{F}_i \cup \{\checkmark\}} \chi_i(x \mid s, a) = 1$$

with the final equality following from $\chi_i(x \mid s, a)$ being defined as a valid distribution on $F_i \cup \{\checkmark\}$. □

Having defined the behaviour of a faulty agent, and observed the validity of our definition, we now proceed to use this to define a faulty system.

Definition 6.7 (Fully Faulty APMAS)**.** *Let $\mathcal{S} = \langle \mathcal{T}, E, \mathcal{V}, V_E \rangle$ be an APMAS, with agents $\mathcal{T} = \{T_1, \ldots, T_k\}$ and valuation functions $\mathcal{V} = \{V_1, \ldots, V_k\}$. Further, let χ be a vector containing for each agent type a fault profile $\chi_i : S_i \times Act_i \to Dist(\mathcal{F}_i \cup \{\checkmark\})$. Then, the fully faulty APMAS $\mathcal{S}^\chi = \langle \mathcal{T}^\chi, E, \mathcal{V}^\chi, V_E \rangle$ is constructed by taking $\mathcal{T}^\chi = \{T_1^{\chi_1}, \ldots, T_k^{\chi_k}\}$ with each $T_i^{\chi_i}$ defined as in Definition 6.6, and $\mathcal{V}^\chi = \{V_1^{\chi_1}, \ldots, V_k^{\chi_k}\}$ with each $V_i^{\chi_i} : S_i^{\chi_i} \times S_E \to \mathcal{P}(AP \cup \{faulty, injected\})$ defined by:*

$$((s, f, e), s_E) \mapsto \begin{cases} \{faulty, injected\} \cup V_i(s, s_E) & \text{if } e = \top \\ \{faulty\} \cup V_i(s, s_E) & \text{if } e = \bot \text{ and } f = \top \\ V_i(s, s_E) & \text{otherwise} \end{cases}$$

Notice that the environment does not exhibit any faults. We add two additional atomic propositions to the language: *faulty* and *injected*, tracking whether an agent has ever exhibited faulty behaviour (and whether it exhibited faulty behaviour at the previous time step, respectively). These atomic propositions allow us to express a number of specifications such as:

$$P^{max}_{\leq 0.5}[G\neg(faulty, (1, 1))]$$

which expresses that the probability of an agent ever exhibiting a fault does not exceed 0.5.

We can also express probabilistic variants of properties often considered in fault-tolerance literature. For instance:

$$P^{max}_{>0.9}[G((injected, (1,1)) \rightarrow F\phi)] \qquad \text{(Recoverability)}$$

expresses that with high probability even if an agent exhibits a fault the system will still satisfy ϕ at some point in the future. This expresses a notion of resilience of the agents in the system with respect to the fault and the specification ϕ. Note that ϕ may depend on the state of other agents or the environment, and thus can express a property of the whole system rather than just of agent $(1, 1)$.

Finally, the new atomic propositions allow us to limit specifications to agents which have not exhibited faulty behaviour. For instance:

$$P^{max}_{>0.9}[G(\neg(faulty, (1,1)) \rightarrow \phi)]$$

expresses that with high probability whenever an agent is not faulty, the formula ϕ is satisfied.

6.2.3 Probabilistically Faulty Systems

The above notion of a faulty system, in which every agent may exhibit faults, is far too restrictive for many real-life scenarios. It is a more typical assumption is safety analysis that only a certain proportion of agents will exhibit faults. To model these scenarios, we introduce a more realistic model, in which each type of agent is associated with a probability that defines how likely it is to *ever* exhibit faults.

We note that while the fully faulty systems described in the previous systems could be defined by simply modifying the agents to introduce faults this is not the case here. Each agent needs to be given an opportunity to establish if it will be faulty or not according to its faultiness probability, and due to the possibly unbounded number of agents we do not known at design-time how many agents will be in the system and thus do not know how many time-steps to wait in order for this to have happened. In order to overcome this, we

define an initialisation phase which is terminated by a global-synchronous action once all agents have chosen if they will exhibit faults or not.

Definition 6.8 (Probabilistically Faulty APMAS)**.** *Let $\mathcal{S} = \langle \mathcal{T}, E, \mathcal{V}, V_E \rangle$ be an APMAS, with agents $\mathcal{T} = \{T_1, \ldots, T_k\}$ and valuation functions $\mathcal{V} = \{V_1, \ldots, V_k\}$. Further, let χ be a vector containing for each agent type a fault profile $\chi_i : S_i \times Act_i \to Dist(\mathcal{F}_i \cup \{\checkmark\})$, and $p \in [0,1]^k$ be a vector of faultiness probabilities for each agent type. Then, the probabilistically faulty APMAS $\mathcal{S}^{\chi,p} = \langle \mathcal{T}^{\chi,p}, E^{\chi,p}, \mathcal{V}^{\chi,p}, V_E^{\chi,p} \rangle$ is defined as follows.*

The faulty agent template are given by $\mathcal{T}^{\chi,p} = \{T_1^{\chi_1,p_1}, \ldots, T_k^{\chi_k,p_k}\}$ with each $T_i^{\chi_i,p_i} = \langle S_i^{\chi_i,p_i}, \iota_i^{\chi_i,p_i}, Act_i^{\chi_i,p_i}, P_i^{\chi_i,p_i}, t_i^{\chi_i,p_i} \rangle$ defined by:

- *$S_i^{\chi_i,p_i} = S_i \cup S_i^{\chi_i} \cup \{\iota^{\chi_i,p_i}, \iota_f, \iota'_f, \iota_n, \iota'_n\}$, defined by considering all the faulty and non-faulty states, as well as five fresh states used for initialisation below.*
- *$\iota_i^{\chi_i,p_i}$, the new initial state.*
- *$Act_i^{\chi_i,p_i} = Act_i \cup \{init, g, g'\}$ where $init \in A$ and $g, g' \in GS$, defined by introducing a fresh asynchronous action init and fresh global-synchronous actions g and g', used for initialisation.*
- *$P_i^{\chi_i,p_i} : S_i^{\chi_i,p_i} \to \mathcal{P}(Act_i^{\chi_i,p_i})$ is given by:*

$$s \mapsto \begin{cases} \{a\} & \text{if } s = \iota_i^{\chi_i,p_i} \\ \{g\} & \text{if } s \in \{\iota_f, \iota_n\} \\ \{g'\} & \text{if } s \in \{\iota'_f, \iota'_n\} \\ P_i(s) & \text{if } s \in S_i \\ P_i^{\chi_i}(s) & \text{if } s \in S_i^{\chi_i} \end{cases}$$

- $t_i^{\chi_i,p_i} : S_i^{\chi_i,p_i} \times Act_i^{\chi_i,p_i} \to Dist(S_i^{\chi_i,p_i})$ *is given by:*

$$t_i^{\chi_i,p_i}(s' \mid s,a) \triangleq \begin{cases} p_i & \text{if } s = \iota_i^{\chi_i,p_i}, a = init, s' = \iota_f \\ 1 - p_i & \text{if } s = \iota_i^{\chi_i,p_i}, a = init, s' = \iota_n \\ 1 & \text{if } s = \iota_f, a = g, s' = \iota'_f \\ & \quad \text{or } s = \iota'_f, a = g', s' = \iota_i^{\chi_i} \\ & \quad \text{or } s = \iota_n, a = g, s' = \iota'_n \\ & \quad \text{or } s = \iota'_n, a = g', s' = \iota_i \\ t_i(s' \mid s,a) & \text{if } s, s' \in S_i \\ t_i^{\chi_i}(s' \mid s,a) & \text{if } s, s' \in S_i^{\chi_i} \\ 0 & \text{otherwise} \end{cases}$$

The environment $E^{\chi,p} = \langle S_E^{\chi,p}, \iota_E^{\chi,p}, Act_E^{\chi,p}, P_E^{\chi,p}, t_E^{\chi,p} \rangle$ *is given by:*

- $S_E^{\chi,p} = S_E \cup \{\iota_E^{\chi,p}, \iota'\}$*, defined by adding two states to* S_E *used for initialisation.*
- $\iota_E^{\chi,p}$*, the new initial state.*
- $Act_E^{\chi,p} = Act_E \cup \{g, g'\}$ *where* $g, g' \in GS$*, defined by adding two fresh global-synchronous actions.*
- $P_E^{\chi,p} : S_E^{\chi,p} \to \mathcal{P}(Act_E^{\chi,p})$ *is given by:*

$$s \mapsto \begin{cases} \{g\} & \text{if } s = \iota_E^{\chi,p} \\ \{g'\} & \text{if } s = \iota' \\ P_E(s) & \text{if } s \in S_E \end{cases}$$

- $t_E^{\chi,p} : S_E^{\chi,p} \times Act_E^{\chi,p} \to Dist(S_E^{\chi,p})$ *is given by:*

$$t_E^{\chi,p}(s' \mid s,a) \triangleq \begin{cases} 1 & \text{if } s = \iota_E^{\chi,p}, a = g, s' = \iota' \\ & \quad \text{or } s = \iota', a = g', s' = \iota_E \\ t_E(s' \mid s,a) & \text{if } s, s' \in S_E \\ 0 & \text{otherwise} \end{cases}$$

The agent valuation functions $\mathcal{V}^{\chi,p} = \{V_1^{\chi_1,p_1}, \ldots, V_k^{\chi_k,p_k}\}$ *with each* $V_i^{\chi_i,p_i} : S_i^{\chi_i,p_i} \times S_E^{\chi,p} \to \mathcal{P}(AP \cup \{starting, faulty, injected\})$ *is defined by:*

$$(s, s_E) \mapsto \begin{cases} V_i^{\chi_i}(s, s_E) & \text{if } s \in S_i^{\chi_i}, s_E \in S_E \\ V_i(s, s_E) & \text{if } s \in S_i, s_E \in S_E \\ \emptyset & \text{otherwise} \end{cases}$$

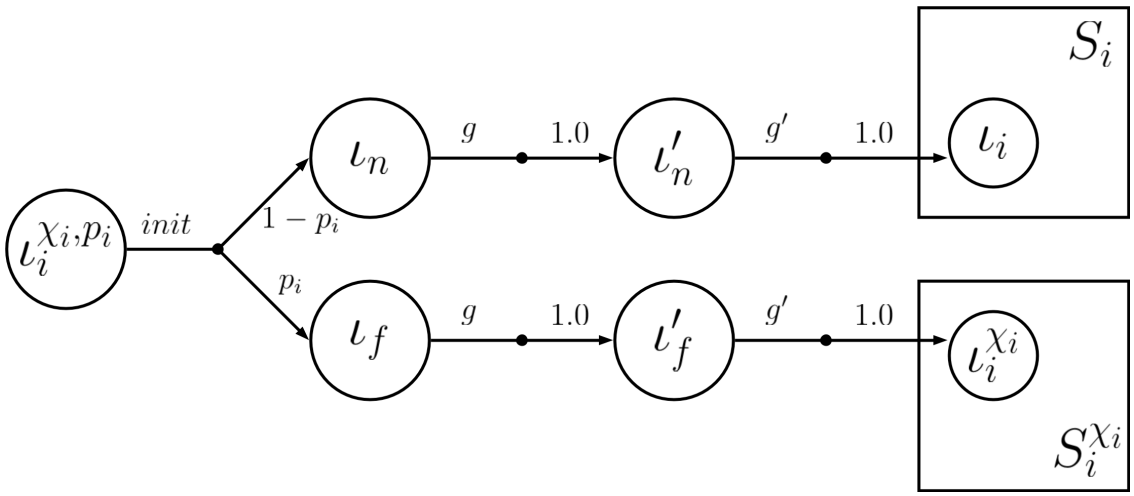
Finally, the environment valuation function $V_E^{\chi,p} : S_E^{\chi,p} \to \mathcal{P}(AP \cup \{starting, faulty, injected\}$ *is defined by:*

$$s_E \mapsto \begin{cases} \{starting\} & \text{if } s_E = \iota' \\ V_E(s_E) & \text{if } s_E \in S_E \\ \emptyset & \text{otherwise} \end{cases}$$

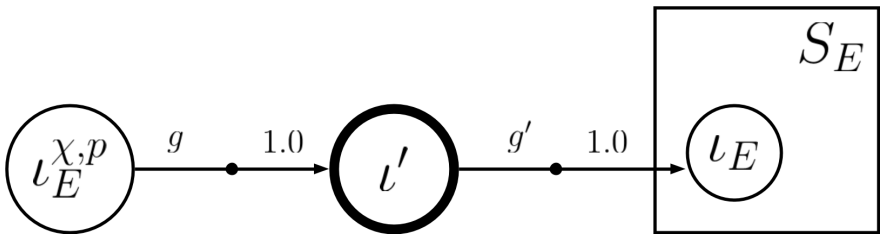
Intuitively, the original system is extended with an initialisation phase where each agent asynchronously uses the *init* transition to determine whether it will be faulty with probability p_i. Following this, the global-synchronous transition g takes the system to a state where our new atomic proposition *starting* holds. Finally, the global-synchronous transition g' starts the system. This initialisation process is depicted in Figure 6.3.

Note that the definition above results in a valid APMAS – the agent transition functions $t_i^{\chi_i,p_i}$ and the environment transition function $t_E^{\chi,p}$ all define valid probability distributions.

Observation 6.2. *Let* $\mathcal{S} = \langle \mathcal{T}, E, \mathcal{V}, V_E \rangle$ *be an APMAS, with agents* $\mathcal{T} = \{T_1, \ldots, T_k\}$ *and valuation functions* $\mathcal{V} = \{V_1, \ldots, V_k\}$*. Further, let* χ *be a vector containing for each agent*



(a) One of the agent templates.



(b) The environment.

Figure 6.3 The APMAS for a probabilistically faulty system. The bold states represent ones where the atomic proposition *starting* holds. The g and g' actions are global-synchronous ones, whilst the *init* action is asynchronous.

type a fault profile $\chi_i : S_i \times Act_i \to Dist(\mathcal{F}_i \cup \{\checkmark\})$*, and* $p \in [0,1]^k$ *be a vector of faultiness probabilities for each agent type. Then, for each agent type* i *we have*

$$\sum_{s' \in S_i^{\chi_i,p_i}} t_i^{\chi_i,p_i}(s' \mid s,a) = 1 \tag{6.1}$$

for all $s \in S_i^{\chi_i,p_i}$ *and* $a \in P_i^{\chi_i,p_i}(s)$*. We also have*

$$\sum_{s'_E \in S_E^{\chi,p}} t_E^{\chi,p}(s'_E \mid s_E, a_E) = 1 \tag{6.2}$$

for all $s_E \in S_E^{\chi,p}$ *and* $a_E \in P_E^{\chi,p}(s_E)$*.*

Proof. For Equation (6.1), we proceed by considering the different possibilities for the state s of the agent. If $s \in \{\iota_f, \iota'_f, \iota_n, \iota'_n\}$ note there is precisely one non-zero transition and it has probability 1. If $s \in S_i$ or $s \in S_i^{\chi_i}$, the result follows from t_i or $t_i^{\chi_i}$ defining a valid probability distribution. Finally, if $s = \iota_i^{\chi_i,p_i}$ note that the two transitions have probability p_i and $1 - p_i$ and therefore sum to 1.

For Equation (6.2), we again proceed by considering different possibilities for the state s_E of the environment. If $s_E \in \{\iota_E^{\chi,p}, \iota'\}$ note there is precisely one non-zero transition and it has probability 1. Finally, if $s_E \in S_E$ the result follows by t_E defining a valid probability distribution. □

Armed with the above definitions, we are interested in assessing whether an unbounded probabilistic system is resilient with respect to a specification when subjected to probabilistically faulty agents. To do so, we formulate the following decision problem. This is an extension of the PMCP in Definition 5.1 to incorporate the notion of faults.

Definition 6.9 (PFTP of PLTL on APMAS)**.** *Let* $\mathcal{S}$ *be an APMAS, and* $\phi = P_{\bowtie x}^{max/min}[\psi]$ *be a PLTL formula. Further, let* χ *be a vector of fault profiles and* p *a vector of faultiness probabilities. Then, the parameterised fault-tolerance problem (PFTP) concerns checking*

$$\mathcal{S}^{\chi,p} \models P_{\bowtie x}^{max/min}[G(starting \to X\psi)]$$

where $\models$ *denotes satisfaction according to Definition 5.1. If this holds, we write* $\mathcal{S} \models_p^{\chi} \phi$*.*

Algorithm 6.2 Decision procedure for the PFTP

Input: APMAS $\mathcal{S}$, PLTL formula $\phi = P^{max/min}_{\bowtie x}[\psi]$, fault profiles χ, faultinesses p

Output: Boolean, or `FAIL`

1: **function** `PFTP`($\mathcal{S}$, ϕ, χ, p)

2: Construct $\mathcal{S}^{\chi}_{p}$ via Definition 6.8

3: **return** `PMCP-Unbounded`($S^{\chi}_{p}, P^{max/min}_{\bowtie x}[G(starting \rightarrow X\psi)]$)

4: **end function**

Definition 6.9 allows us to recast the problem of checking fault tolerance as a simpler parameterised model checking query for the amended system under a revised specification. We note that this problem is an extension of Definition 5.1, which was already undecidable. Thus, it is also undecidable in general.

So, to check whether a system $\mathcal{S}$ satisfies a path formula ψ under the fault profiles χ and with the probabilities of agents being faulty given by p, we instead check the formula $G(starting \rightarrow X\psi)$ in the transformed system $\mathcal{S}^{\chi,p}$. This corresponds to what we would intuitively expect, since we wish to check that ψ holds after we have completed the initialisation phase where each agent chooses whether to behave in a faulty manner.

Given the above, we can introduce a simple procedure for checking the PFTP. This is presented in Algorithm 6.2. Note that this procedure is incomplete, since it relies on Algorithm 5.1, which is itself incomplete.

6.3 Summary

This chapter has explored two possible extensions to our models and specifications. In the first half, we considered the PMCP of P[ATL*] on SPMAS (Definition 6.1). We showed that the same abstract model we previously considered can also be used to find bounds on both the maximal probability a coalition of agents can achieve (Lemmas 6.1 and 6.2) and the minimal probability (Lemmas 6.3 and 6.4). We combined these results to show that our

verification procedure (Algorithm 6.1) is sound (Theorem 6.1).

In the second half of the chapter, we considered methods for analysing the behaviour of systems that may exhibit faults. To achieve this, we defined both fully faulty systems in which all agents may exhibit faults (Definition 6.7) and probabilistically faulty systems in which there is a fixed probability that an agent will exhibit faults (Definition 6.8). We defined the problem of verifying such a faulty system (Definition 6.9) and gave a procedure for solving this verification problem using our previous techniques (Algorithm 6.2).

Chapter Seven

Implementation and Evaluation

In this chapter, we present our implementation of some of the techniques described earlier. Further, we evaluate our implementation against three case studies and report some experimental results, in order to check the usability and scalability of our techniques.

The material presented here is drawn from the experimental sections of all the papers discussed in the previous chapters [Lomuscio and Pirovano, 2018, 2019, 2020a,b], with the important distinction that we have combined all the separate implementations previously developed into one complete toolkit. Additionally, the toolkit has been extended to support models with multiple different agent templates rather than just the single agent templates that were supported in our conference papers.

7.1 Implementation Details

We implemented the techniques described in this thesis into a Java toolkit called `PSV` (**P**robabilistic **S**warm **V**erifier). The source codes for both the toolkit and the case studies reported below are released as open source.[1]

The underlying model checking procedures are provided by two different toolkits depending on the types of properties being considered as no one toolkit supports all the properties we wish to consider. In particular, when considering strategic properties, the

[1] They are available here: https://github.com/edoardopirovano/psv

procedure is implemented by an extension of PRISM-games to handle concurrent stochastic games [Kwiatkowska et al., 2018a]. The other model checking procedures come from PRISM 4.0 [Kwiatkowska et al., 2011].

The toolkit takes as input a description of the behaviour of one or more agent templates and of the environment. The language used to describe the system is inspired by that used in PRISM (which we also re-used some of the parsing procedures from), suitably extended to handle parameterised systems. Further, the toolkit can take a description of faults that may occur, and finally a list of specifications to check the system against.

Depending on the arguments passed to the toolkit, `PSV` will first inject faults into the agent template (if a description of possible faults was provided). Then, it will either construct the abstract model (according to Definition 4.2 if the model is an SPMAS, or Definition 5.2 if the model is an APMAS) or the concrete model of a given number of agents (according to Definition 3.5 if the model is an SPMAS, or Definition 3.13 if the model is an APMAS). It then passes the model constructed to either PRISM-games or PRISM (depending on the property under consideration) to verify the specifications given against it. In particular, our classes implement PRISM's ModelGenerator API[2] which PRISM calls to obtain lists of transitions and resulting states in order to explore the model.

`PSV` can also be passed an additional argument to export the full model constructed to a Graphviz DOT file in order to visualise it using a suitable viewer.[3]

7.1.1 Modelling SPMAS

We write models for SPMAS in a "synchronous swarm file", and by convention use the extension `.ssf` for these files. An example of one of these files can be seen in Figure 7.1.

The code is composed of one or more agent templates, followed by an environment. Note that the environment differs from the agent templates in that there will only ever be one

[2] https://github.com/prismmodelchecker/prism/blob/master/prism/src/prism/ModelGenerator.java

[3] We recommend `xdot` for this: https://pypi.org/project/xdot/

copy of the environment in concrete systems, whereas there may be arbitrarily many copies of each agent template. However, the syntax used to define the two is identical.

An agent template or environment definition begins with a number of variable declarations using the syntax

$$\texttt{v}_i \ : \ \texttt{t}_i \ \texttt{init} \ k_i;$$

which declares a new variable $\texttt{v}_i$ of type $\texttt{t}_i$ with initial value $\texttt{k}_i$. The variable types supported are the same as those in PRISM – they can either be Booleans (declared with `bool`), unbounded integers (declared with `int`) or integers bounded between a and b (declared with `[a, b]`). Note that while unbounded integers can be used for convenience, in order for `PSV` to terminate it must be the case that the part of the model that is actually explored is finite.

After the variable definitions, there are one or more statements of the form

```
[actionName] (guard);
```

which declares that the action `actionName` is enabled in states that satisfy `guard`. Again, the types of guard supported are exactly the expressions supported by PRISM including a number of comparison operators, connectives, etc.[4]

Following the declarations of where actions are enabled, there is a number of updates of the form

$$\texttt{(actionName, guard, } \{a_1, \ldots, a_k\}\texttt{) -> p}_1\texttt{:(update}_1\texttt{) + ... + p}_n\texttt{:(update}_n\texttt{);}$$

These updates are interpreted as follows: When performing `actionName` from a state satisfying `guard`, the update is triggered if the set of actions performed by the agents and the environment contains all of the actions $\{a_1, \ldots, a_k\}$. It is also possible to write ! before the set, which will require instead that none of the actions $\{a_1, \ldots, a_k\}$ are performed in order to trigger the update. If this update is triggered, then there is a probability $\texttt{p}_i$ of $\texttt{update}_i$ being performed for each i in order to transition to the next state. Note that it must be the case that $\Sigma_{i \in \{1,\ldots,n\}} \texttt{p}_i = 1$.

[4]Details can be found at https://www.prismmodelchecker.org/manual/ThePRISMLanguage/Expressions

```
agent
        stateA : [0..1] init 0;

        [a] (stateA=0);
        [b] (stateA=1);

        update
                (a, true, {}) -> 0.5:(stateA'=0) + 0.5:(stateA'=1);
        endupdate
endagent

agent
        stateB : [2..3] init 2;

        [c] (stateB=2);
        [d] (stateB=3);

        update
                (c, true, {b}) -> 1.0:(stateB'=3);
        endupdate
endagent

environment
        state : [4..5] init 4;

        [e] true;

        update
                (e, state=4, {d}) -> 1.0:(state'=5);
        endupdate
endenvironment
```

Figure 7.1 An example of PSV code for an SPMAS. In particular, this code models the system in Figure 3.1.

```
asynchronous      = {a}
agentEnvironment  = {e}
globalSynchronous = {g}

agent module AgentA
  stateA : [1..2] init 1;
  [a] (stateA=1) -> 0.5:(stateA'=1) + 0.5:(stateA'=2);
  [g] (stateA=2) -> 1.0:(stateA'=2);
endmodule

agent module AgentB
  stateB : [3..4] init 3;
  [e] (stateB=3) -> 0.5:(stateB'=3) + 0.5:(stateB'=4);
  [g] (stateB=4) -> 1.0:(stateB'=4);
endmodule

environment module Environment
  stateE : int init 5;
  [e] (stateE=5) -> 1.0:(stateE'=5);
  [g] (stateE=5) -> 0.5:(stateE'=5) + 0.5:(stateE'=6);
  [g] (stateE=6) -> 1.0:(stateE'=6);
endmodule
```

Figure 7.2 An example of `PSV` code for an APMAS. In particular, this code models the system in Figure 3.3.

7.1.2 Modelling APMAS

We write models for APMAS in an "asynchronous swarm file", and by convention use the extension `.asf` for these files. An example of one these files can be seen in Figure 7.2.

The code begins with three sets that define the type of each action (asynchronous, agent-environment or global synchronous). Every action used later in the model must appear in one of these sets in order to specify what synchronisation is required when performing the action. If this is not the case, an exception will be thrown during construction of the model.

This is followed by a number of agent modules. Each agent module defines a number of variables using the same syntax as for SPMAS. These variable definitions are followed by a number of update actions, which use a different syntax from the ones in SPMAS. In

```
agent
(g, stateA=2) -> 0.4:(stateA'=1) + 0.6:true;

agent
(g, stateB=4) -> 0.2:(stateB'=3) + 0.8:true;
```

Figure 7.3 An example of `PSV` code for describing faults. This extends the model in Figure 7.2 to introduce a faults that may send agents back to the starting state when performing the g action, with probability 0.4 and 0.2 respectively for the two agent types.

particular an update action for an APMAS is of the form

$$\texttt{[actionName] (guard) -> p}_1\texttt{:(update}_1\texttt{) + ... + p}_n\texttt{:(update}_n\texttt{);}$$

These update actions are interpreted as follows: If `guard` evaluates to `true` in the current state, then the agent may perform action `actionName`. Upon performing this action, there is a probability $\texttt{p}_i$ of $\texttt{update}_i$ being performed for each i in order to transition to the next state. Note that it must be the case that $\Sigma_{i\in\{1,\ldots,n\}}\texttt{p}_i = 1$.

The agent modules are followed by an environment module which is defined in exactly the same way as an agent (but is distinct in what synchronisation it gives rise to, and in the fact that there may only be one copy of it).

7.1.3 Modelling Faults

Faults for systems are specified in a "fault file" (by convention, using the extension `.ff`). An example of one of these files can be seen in Figure 7.3.

A fault file contains for each agent template the keyword `agent` followed by zero or more faults for that agent template. Note that the agents should appear in the same order in the fault file as they do in the corresponding swarm file. The faults for agents are of the form

$$\texttt{(actionName, guard) -> p}_1\texttt{:(faultyUpdate}_1\texttt{) + ... + p}_n\texttt{:(faultyUpdate}_n\texttt{);}$$

This fault is interpreted as follows: When performing action `actionName` from a state satisfying `guard`, then with probability p_i the update $\texttt{faultyUpdate}_i$ will occur. While we

do require $\Sigma_{i\in\{1,\ldots,n\}}\mathtt{p}_i = 1$ for simplicity, it is possible to model the ✓ case where no fault occurs by simply having `true` (i.e. not changing anything) in one of the updates.

7.1.4 Specifying Properties

We first note that at the bottom of either type of swarm file, it is possible to add one or more labels using the syntax

```
label "name" = expr;
```

which defines a new label called `name` and corresponding to an expression `expr`. Within the expression, to refer to variable `v` of the i-th agent of type j we use `v_j_i`. To refer to a variable `k` of the environment we use `k_E`. For instance, in the example from Figure 7.2, we could write

```
label "firstAgentTransitioned" = (stateA_1_1 = 2);
```

to label the states where the agent $(1, 1)$ has transitioned to state 2. These labels can be used in the properties file which, by convention, we use the extension `.prop` for. This file must contain a sequence of properties expressed in the desired logic, such as

```
P<=0.9 [ F<4 ("firstAgentTransitioned") ]
```

which expresses that the probability of the first agent transitioning within four time-steps is at most 0.9.

For non-strategic properties, the full syntax of properties that can be expressed in the properties file is those supported by PRISM[5] with the exception of the reward operator, which is not considered here.

For strategic properties, `PSV` supports the same syntax as PRISM-games,[6] once again with the exception of reward-based properties. When specifying coalitions, the identifier `agent_j_i` can be used to refer to the i-th agent of type j, and `env` can be used to refer to the environment.

[5]See https://www.prismmodelchecker.org/manual/PropertySpecification/SyntaxAndSemantics

[6]See https://www.prismmodelchecker.org/games/properties.php

7.2 Case Studies

We now describe three case studies that can be analysed by using PSV. All timing results were obtained on a machine running Ubuntu 20.04 (Linux kernel 5.4.0) with an Intel i9-9940X processor and 128GB of RAM. The code was run using OpenJDK 15 (64-bit version), with 96GB of RAM allocated to the Java heap.

We do not provide a comparison of our toolkit to any others as no other toolkit provides a way of checking unbounded probabilistic multi-agent systems as we do here. The checking of the concrete and abstract models that we build is handled by PRISM and PRISM-games, both of which have already been extensively benchmarked elsewhere [Kwiatkowska et al., 2012, 2018b].

7.2.1 Autonomous Robots

As a first case study to verify the applicability of PSV we consider a probabilistic variant of the autonomous robots example from [Fagin et al., 1995]. In this scenario, there are a number of robots moving along a track of infinite length. The robots begin at the start of the track and are moved synchronously along by an environment until they decide to stop. The robots aim to stop in a target region (for our experiment, we fixed this to be between 19 and 21 units of distance along the track). Some of the robots are equipped with sensors to detect their position, but others are not and must rely on communication from robots with a sensor in order to decide when to stop.

For our experiment, we assumed that the sensor of the robots equipped with one is noisy and can give readings up to two below or up to two above the actual position. Further, we assumed that all these readings are equally likely (i.e., they all have a probability of 0.2). The robots with a sensor decide to halt if the reading they receive from their sensor is at least 20. When they halt, they also broadcast a signal to sensorless agents telling them to stop. Our model is an APMAS and includes a combination of asynchronous, agent-environment

```
...
agent module WithoutSensor
        location : [1..22] init 1;
        stopped : bool init false;
        [move] (!stopped) -> 1.0:(location'=location+1);
        [signal] (true) -> 1.0:(stopped'=true);
endmodule
...
```

Figure 7.4 A snippet of the APMAS code modelling the autonomous robots scenario giving the code for sensorless robots.

and global-synchronous actions. Part of the code for the model can be seen in Figure 7.4.

We considered the $(0, 1)$-indexed PLTL property

$$P_{\leq p}^{max}[F((stopped, (2, 1)) \wedge \neg(target, (2, 1)))]$$

where *stopped* is an atomic proposition that holds when an agent has halted, and *target* holds when an agent is within the target region. The property expresses that we are certain that the probability of a sensorless agent stopping outside the target region is at most p. We checked this for a different number of concrete agents with sensors, and recorded the largest p for which this holds in each case. The results are shown in Figure 7.5.

We see that when the number of agents with sensors is increased the probability also increases. This is as expected, since when the number of agents is increased it is more likely that at least one of the agents with a sensor will misjudge the stopping position and start broadcasting a signal too early, causing our sensorless agent to stop early. Further, note that the maximum probability computed by the abstract model is 1. This corresponds to our intuition that when we have arbitrarily many agents the probability of at least one misjudging the position will tend to 1.

The abstract model has 3,352 states and 21,146 transitions. It takes `PSV` around 0.24 seconds to construct this model and pass it to PRISM, which then needs around 0.08 seconds to compute the value of p. The largest of the concrete models (with 7 concrete agents) has 15,676,417 states and 99,246,301 transitions. It takes around 164 seconds for our tool to

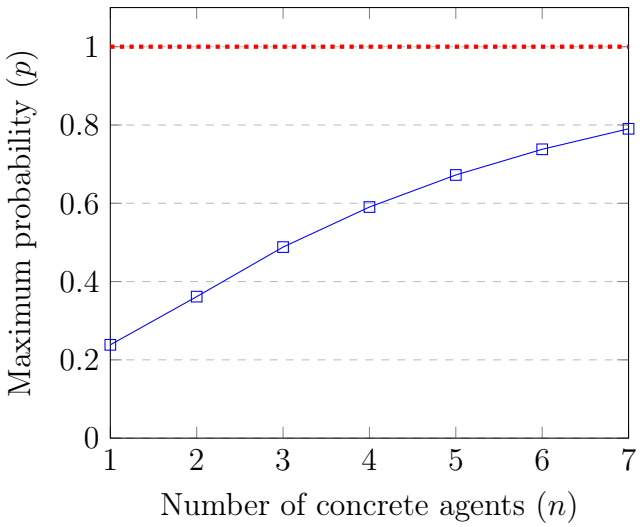


Figure 7.5 For different numbers of concrete type 1 agents (robots with a sensor), the maximum value of p for which the property $P^{max}_{\leq p}[F((stopped, (2,1)) \wedge \neg(target, (2,1)))]$ holds, i.e. how likely it is for a sensorless robot to stop outside the target region. The red line shows the maximum value computed by the abstract model.

construct the model and pass it to PRISM which then computes p in around 278 seconds.

7.2.2 Foraging Protocol

For a richer example, we also modelled a foraging protocol [Campo and Dorigo, 2007; Liu and Winfield, 2010]. In this scenario, agents begin resting in a nest. They may choose to leave the nest and search for food, which they then retrieve and bring back to the nest. Upon returning to the nest the agents return to the resting state.

We considered agents of two types. Both types of agents choose to stop resting with probability 0.5. However, when searching for food the first type of agent will look up to two units of distance away whereas the second will only look one unit of distance away. Accordingly, we gave the second type of agent a probability of 0.15 of finding food when searching for it, and the first a higher probability of 0.3 (with a 0.15 chance it is one unit of distance away, and a 0.15 chance it is two). Both types of agent immediately travel to the food and bring it back to the nest upon locating it.

```
...
agent module Agent1
        state : [0..3] init 0;
        fromHome : [0..2] init 0;
        ...
        [move] state=3 & fromHome>0 -> 1.0:(fromHome'=fromHome-1);
        [deposit] state=3 & fromHome=0 -> 1.0:(state'=0);
endmodule
...
environment module Env
        deposited : [0..2] init 0;
        [deposit] deposited<2 -> 1.0:(deposited'=deposited+1);
endmodule
```

Figure 7.6 A snippet of the code for the APMAS modelling the foraging scenario giving the part of the model that encodes robots moving back to the nest with food and depositing it.

Our model for this scenario is an APMAS involving a combination of asynchronous and agent-environment actions. Asynchronous actions are used to model the agents moving, searching for food, and collecting the food. Agent-environment actions are used to model the agents depositing food in the nest (which is captured by the environment). Part of the code for our model can be seen in Figure 7.6.

We used `PSV` to investigate the probability that two units of food will be found and deposited within a certain number of time-steps. In particular, we checked for different values of p and k the $(0,0)$-indexed PLTL_k property

$$P^{\max}_{\leq p}[F^{<k}\mathrm{deposited}_2],$$

where $\mathrm{deposited}_2$ is an atomic proposition that holds when two units of food have been deposited in the nest by any agent. The results of checking this property for different values of p and k against the abstract model are recorded in Table 7.1.

The results match our expectations: when the agents are given a longer number of time-steps to collect food then there is a higher probability that they will succeed. The abstract model constructed to verify the properties takes around 46 seconds to build and has

		k							
		8	9	10	11	12	13	14	15
	0.25	True	False	False	False	False	False	False	False
	0.50	True	True	False	False	False	False	False	False
	0.75	True	True	True	False	False	False	False	False
	0.80	True	True	True	False	False	False	False	False
p	0.85	True	True	True	False	False	False	False	False
	0.90	True	True	True	True	False	False	False	False
	0.95	True	True	True	True	True	False	False	False
	0.98	True	True	True	True	True	True	False	False
	0.99	True	True	True	True	True	True	True	False

Table 7.1 For different numbers of time-steps k and probabilities p, whether or not the property $P^{max}_{\leq p}[F^{<k}\text{deposited}_2]$ holds in the abstract model.

259,560 states and 6,684,624 transitions. Once the abstract model is constructed, checking the individual properties on it takes a negligible time (around 200ms per property).

As a further analysis, we inject one fault into the system encoding the fact that whenever a robot tries to move and is carrying food, it may drop the food with some probability p_f. If it does this, it will return to the nest with no food. After fault injection, we study once again specifications of the form $P^{\max}_{\leq p}[F^{<k}\text{deposited}_2]$.

Intuitively the probabilities p for which this specification will hold depends on the fault probability p_f. By using `PSV` we can ascertain this relation precisely; we conducted this analysis and recorded in Figure 7.7 the maximum p for which the property holds for different values of k and p_f. The abstract model that `PSV` generated had $727,828$ states and $21,628,009$ transitions. Injecting the faults was instantaneous, constructing the model took approximately 187 seconds and each specification was checked in approximately 1 second.

As would be expected, increasing the probability p_f that agents will exhibit a fault and

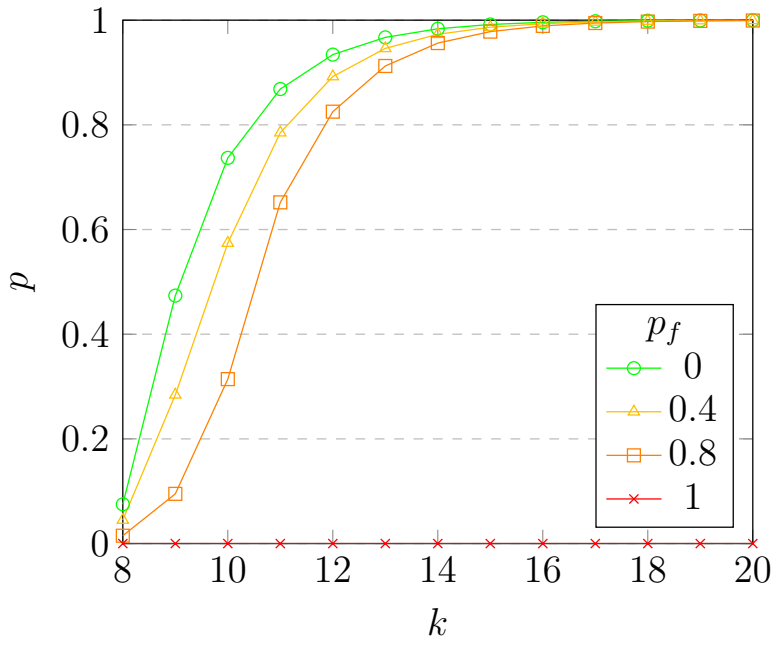


Figure 7.7 For different fault probabilities p_f and time-steps k, the maximum value of p for which $P^{\max}_{\leq p}[F^{<k}\text{deposited}_2]$ holds in the abstract model.

drop the food they are carrying reduces their likelihood of success. Reassuringly, however, when given enough time (i.e. as k gets larger) the probability that the agents will manage to bring two units of food back to the nest still converges to 1.

7.2.3 Channel Jamming Scenario

For our final case study, we modelled a channel jamming security protocol [Zhu et al., 2010]. In our model of the protocol, there are k channels available to agents to send messages. At each time-step, each agent can choose one channel to send a message. A number of agents in the system are attackers; each of them can jam a channel. It is assumed that if a message is sent along a non-jammed channel, then it is successfully transmitted with probability 0.4. If it is sent along a channel that is jammed by at least one attacker then this probability drops to 0.1.

The sending of messages and jamming of channels is modelled in an SPMAS, with the choices of action for agents at each time-step being to send one message along a channel i

```
agent
  transmitted : bool init false;

  [transmit] (transmitted=true);

  [block0] true;
  [block1] true;

  [transmit0] (transmitted=false);
  [transmit1] (transmitted=false);

  update
    (transmit0, true, !{block0}) -> 0.4:(transmitted'=true);
    (transmit0, true, {block0}) -> 0.1:(transmitted'=true);

    (transmit1, true, !{block1}) -> 0.4:(transmitted'=true);
    (transmit1, true, {block1}) -> 0.1:(transmitted'=true);

    (transmit, true, {}) -> 1.0:(transmitted'=false);
  endupdate
endagent

environment
  received : [0..3] init 0;

  [receive] true;

  update
    (receive, received<3, {transmit}) -> 1.0:(received'=received+1);
  endupdate
endenvironment
```

Figure 7.8 The code for the SPMAS modelling the jamming scenario in the case with two channels and three messages.

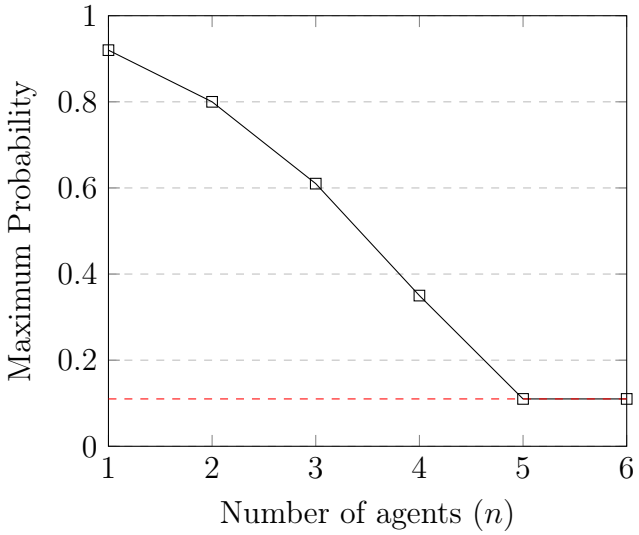


Figure 7.9 Graph showing the maximum value of p for which the property $\langle\langle(1,1),E\rangle\rangle P_{\geq p}[F^{\leq 15} received_3]$ holds for different numbers n of concrete agents. The number of channels is fixed to 4. The red dashed line shows the expected lower bound computed by the abstract model.

(with $0 \leq i < k$) or block a channel i (with $0 \leq i < k$). The environment acts as a receiver for the messages, and tracks how many messages have been received. The code for the model with two channels and three messages being transmitted can be seen in Figure 7.8.

We wish to verify the property that with probability p the first agent can ensure that at least i messages are transmitted within j time-steps. This can be expressed by the (1)-indexed P[ATL*] property

$$\langle\langle(1,1),E\rangle\rangle P_{\geq p}[F^{\leq j} received_i]$$

where $received_i$ is an atomic proposition that holds after i messages have been received.

For our first experiment, we fixed the number of channels available to 4, the number of messages being transmitted to 3, and the number of time-steps allowed for transmission to 15. We then used our tool to compute the lower bound for our property as the number of agents n varies (as given by Lemma 6.2, using the abstract model). We also computed the actual value of this for different numbers of agents n. Our results are shown in Figure 7.9.

As expected, the actual values are above the calculated lower bound. Further, the mini-

mum value is attained once there are at least 5 agents in the system. This is expected since this corresponds to the system where there are 4 attackers, which is enough for them to have a joint strategy to block every channel. The computation of the minimum value using the abstract model is much more efficient than constructing systems of increasing size to find this; constructing and verifying the abstraction takes around 4 seconds, whilst constructing and checking the system of size 5 takes around 567 seconds.

As a further experiment to verify the scalability of our tool, we checked the time taken to construct the abstract model and use this to compute the maximum value of p for which $\langle\langle(1,1),E\rangle\rangle P_{\geq p}[F^{\leq 15} received_i]$ holds as we varied the number of messages that we wished to receive and the number of channels. Our results are in Table 7.2, along with the total number of states and total number of transitions in the abstract model.

Note that varying the number of channels does not change the probability of success of our transmitting agent. This is as expected, since in our abstract model we assume the worst case where there may be arbitrarily many opposing agents. Therefore, all the channels will always be blocked and increasing the number of them does not benefit our agent. Increasing the number of messages that our agent needs to transmit reduces its chances of success.

Further, notice that varying the number of channels changes only the number of transitions in the model since there are more choices of channel to transmit on but the number of messages we have to keep track of remains unchanged. Varying the number of messages being sent, on the other hand, also changes the number of states.

7.3 Summary

In this chapter, we have presented our Java implementation of the techniques previously outlined which we named `PSV`. The code for both our toolkit and the case studies analysed are released as open-source and our experiments are easily reproducible on a Linux workstation. As we rely on PRISM and PRISM-games for the underlying model checking procedures, we

		k		
		3	4	5
i	5	0.999 24 states 14,976 transitions 0.34 sec + 0.23 sec	0.999 24 states 78,336 transitions 3.53 sec + 1.56 sec	0.999 24 states 387,072 transitions 94.0 sec + 34.0 sec
	10	0.903 44 states 27,456 transitions 0.57 sec + 0.35 sec	0.903 44 states 143,616 transitions 5.97 sec + 2.81 sec	0.903 44 states 709,632 transitions 172 sec + 55.0 sec
	15	0.374 64 states 39,936 transitions 0.67 sec + 0.50 sec	0.374 64 states 208,896 transitions 8.50 sec + 3.92 sec	0.374 64 states 1,032,192 transitions 258 sec + 90.2 sec
	20	0.034 84 states 52,416 transitions 0.81 sec + 0.65 sec	0.034 84 states 274,176 transitions 10.9 sec + 5.33 sec	0.034 84 states 1,354,752 transitions 335 sec + 101 sec
	25	0.001 104 states 64,896 transitions 0.94 sec + 0.77 sec	0.001 104 states 339,456 transitions 13.8 sec + 6.30 sec	0.001 104 states 1,677,312 transitions 419 sec + 146 sec

Table 7.2 For different values of k (available channels) and i (number of messages to transmit), the maximum value of p for which $\langle\langle(1,1),E\rangle\rangle P_{\geq p}[F^{\leq 15} received_3]$ holds in the abstract model, the number of states and transitions in this model, and the time needed to respectively build the model and compute the value.

note that future improvements to these will improve the times required to verify properties against our systems.

Nonetheless, construction of the abstract models for both SPMAS and APMAS is very time consuming and limits the scalability of our toolkit. To some extent, this is inevitable as we showed in Observation 4.4 and Observation 5.3 that these models can grow exponentially in the number of reachable states in an agent template.

However, one limitation of `PSV` is that models are completely constructed before being passed to PRISM or PRISM-games for checking. More sophisticated approaches that build the model on-the-fly as it is required by the verification procedure could also be explored, and these may improve the scalability of our tool in some cases.

Our toolkit is highly expressive; it supports two types of models, three specification logics, and fault injection techniques. However, there are certainly areas where more expressivity could be explored. For instance, we have not studied reward-based properties that are often considered in the probabilistic model checking literature. We survey in more detail some possible avenues to increase the expressivity of our toolkit in Section 8.3.3.

Chapter Eight

Conclusions

This thesis has tackled the verification of unbounded probabilistic multi-agent systems, thus making an initial contribution to combining parameterised model checking with probabilistic techniques.

8.1 Summary of Contributions

In this thesis we have achieved the main objectives we set out in Section 1.1. In particular, we have defined two semantics for reasoning about unbounded probabilistic multi-agent systems which we named SPMAS and APMAS. We went on to define three logics for specifying properties of these systems (PLTL, PLTL$_k$, and P[ATL*]) and we studied the parameterised model checking problems that these logics give rise to. In particular, we gave a complete decision procedure for one problem and an incomplete decision procedure for two others. We also presented a method for injecting faults into APMASs. These theoretical results are summarised in Table 8.1.

On the practical front, we have implemented our procedures into the toolkit `PSV`, and used this to study three practical case studies from a variety of different application domains: the autonomous robots example, a foraging protocol, and a channel jamming scenario. All our code is released as open-source, and the experiments presented in this thesis can be

	PMCP with properties in			
	PLTL	PLTL_k	P[ATL*]	Fault Injection
SPMAS		✔	✓	
APMAS	✓			✔

Table 8.1 A summary of the decision problems we have considered on our two models for UPMAS. A light checkmark (✓) denotes that we have developed an incomplete decision procedure, whilst a bold checkmark (✔) denotes that we have developed a complete procedure. Blank cells indicate we did not study the problem.

reproduced on a standard workstation.

8.2 Comparison with Other Approaches

As we previously discussed in Chapter 2, extensive work has been carried out in both parameterised and probabilistic model checking with the former enabling the verification of unbounded multi-agent systems and the latter of stochastic systems. We do not cover this background again here, and instead highlight some differences between the work presented in this thesis and alternative approaches.

The parameterised verification of probabilistic systems is a relatively unexplored area. In [Graham et al., 2009], the authours present a technique for verifying probabilistic network protocols that is based on identifying a sub-class of systems (named *degenerative*) in which larger networks eventually behave like smaller ones. The approach in this thesis tackles a similar problem but instead of studying a sub-class of systems, we here aimed to give a more general verification technique for a broader ranger of systems.

In [Bertrand and Fournier, 2013], a parameterised semantics is introduced that includes both probabilities and continuous time. Reachability properties are studied on both systems where the number of participants is fixed and ones where participants may enter and leave at run-time. Some of the resulting decision problems are found to be decidable whilst others are

shown to be undecidable. The models considered there are distinct from ours. In particular, we do not have a continuous notion of time and the communication pattern between agents in our systems is different from the broadcasting and receiving pattern used there which is more suited to network protocols than the AI applications we considered here. Further, the specification languages we have considered in thesis are more expressive than the qualitative reachability ones considered there.

Model checking approaches like the one presented in this thesis are not the only technique that has been considered to verify properties of multi-agent systems. Another formalism that has been widely studied is that of Petri nets [Reisig, 1985] or equivalently vector addition systems [Rackoff, 1978]. This work has included analysis of reachability properties [Leroux and Schmitz, 2015] and techniques to identify dynamic cut-offs [Kaiser et al., 2010; Spalazzi and Spegni, 2020]. Extensions have also been developed that incorporate continuous time [Penczek and Pólrola, 2004; Abdulla and Nylén, 2001; Jacobsen et al., 2011]. We note that encoding a system into a Petri net requires significant input from an expert, whereas the model checking approach we have presented aims to verify systems in a more automated manner.

A further formalism that has been used to analyse multi-agent systems is that of population protocols [Angluin et al., 2004]. In this semantics, agents interact in a pair-wise manner and update their state according to the state of the agent they are interacting with. A protocol is well-defined, and said to compute a predicate P if all fair sequences of interactions from an initial configuration C eventually result in the agents agreeing on a value $P(C)$. It has been proved [Angluin et al., 2006] that the predicates computable by population protocols are exactly those definable in Presburger arithmetic [Ginsburg and Spanier, 1966]. Further, it has also been shown [Esparza et al., 2017] that the problem of checking whether a protocol is well-defined and computes a given predicate is decidable by reduction to reachability in Petri nets. As with Petri nets, significant human input is needed to encode a system into a population protocol. This limits the applicability of the technique.

There are also a variety of other methods that have been proposed to certify the behaviour of multi-agents that are less closely related to the verification approaches considered here, including techniques such as process calculi [Meyer, 2009] and rate equations [Lerman and Galstyan, 2002]. As with the other techniques described, some expert input is required to define the models used in these systems. Further, the properties considered with these techniques are typically of the whole population and properties of individual agents cannot be checked as we have done in this thesis.

8.3 Future Work

The combination of parameterised model checking with probabilistic systems is a novel area. Accordingly, there are still many possible avenues for further work. We outline some in this section.

8.3.1 Remaining Decision Problems

One fairly natural extension of this work would be to fill in the gaps in Table 8.1 by studying the remaining decision problems. We briefly outline here some thoughts on how each of these could be tackled:

- **PMCP of SPMAS against PLTL:** A partial decision procedure for this could be developed as was done in Chapter 5 but using the abstract model for SPMAS instead of the one for APMAS. We did not pursue this as many of the results would be very similar to the ones we already obtained.

- **PMCP of APMAS against PLTL$_k$:** Similarly, following the same process as in Chapter 4 but substituting APMAS for SPMAS should allow a procedure for this decision problem to be developed.

- **PMCP of APMAS against P[ATL*]:** Here, defining the decision problem would prove difficult. Notice that in a synchronous system, every agent is making a choice of action at each step, so it makes sense to define joint strategies as combining the choices of each agent. This allowed us to define the decision problem of SPMAS against P[ATL*]. However, in asynchronous systems like APMAS there can be more than one agent able to perform an asynchronous action in each state. This means that we also need a way to reason about which agent is given the opportunity to act; this would have to be a part of how the decision problem is defined.

- **Fault Injection on SPMAS:** While we illustrated fault injection on APMAS, the same definitions with some minor tweaks could also be used for SPMAS. This would allow a definition for fully faulty systems. Further, defining a probabilistically faulty system is simpler on SPMAS, as instead of an initialization phase like the one illustrated in Figure 6.3, one can simply use the first action (which all agents perform simultaneously in an SPMAS) to decide whether each agent is faulty or not.

8.3.2 Scalability and Applications

The case studies presented in Section 7.2 allowed us to verify the functioning of `PSV` and demonstrate the potential usefulness of our techniques. However, there are still limitations in the size of the systems that can verified. In particular, because our procedure is exponential in the number of states that agents can be in, it is computationally infeasible to verify scenarios with many reachable agent states.

In order to improve the applicability of our toolkit to modelling richer scenarios, we would need to develop a more scalable algorithm. One way this could be achieved is by identifying a subclass of APMAS or SPMAS that can be verified in a more efficient manner.

8.3.3 Increased Expressivity

A final possible avenue for future work is expanding the number of situations that can be modelled by introducing more expressive models and/or considering richer specification logics. Some potential options could include:

- **Reward properties:** A number of probabilistic logics, such as PCTL [Kwiatkowska et al., 2007] and rPATL [Chen et al., 2013], include a way of reasoning about rewards. In particular, when using these logics the models and specifications under consideration are augmented with numerical rewards that are obtained by visiting states. We did not pursue reward-based properties in this thesis, but it would be fairly natural to extend our analysis to cover these.

- **Probabilistic knowledge:** Logics have been proposed to reason about knowledge in a probabilistic setting [Huang and Luo, 2013; Moses and Zamir, 2020]. It would be interesting to study the parameterised model checking problem of these logics against models such as the SPMAS and APMAS ones we have developed here. To the best of our knowledge, no work has ever been carried out in the overlap of parameterised model checking with probabilistic knowledge.

- **Open systems:** Most work on verification of multi-agent systems, including this thesis, focusses on systems in which the number of participants does not change during the course of a run. There are, however, some formalisms which consider open systems, in which agents can join and leave the system at run-time [Belardinelli et al., 2015; Kouvaros et al., 2019]. It should be possible to extended some of the results in this thesis to study open systems.

- **Continuous time:** One could consider extending work in this thesis to reason about systems that are both probabilistic and have a continuous notion of time [Norman et al., 2013], rather than the discrete notion of time considered here.

References

Abdulla, P. A. and Nylén, A. (2001). Timed Petri Nets and BQOs. In *Proceedings of the 22nd International Conference on Applications and Theory of Petri Nets (ICATPN01)*, volume 2075 of *Lecture Notes in Computer Science*, pages 53–70. Springer.

Aminof, B., Rubin, S., Stoilkovska, I., Widder, J., and Zuleger, F. (2018). Parameterized model checking of synchronous distributed algorithms by abstraction. In *Proceedings of the 19th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI18)*, volume 10747 of *Lecture Notes in Computer Science*, pages 1–24. Springer.

Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M., and Peralta, R. (2004). Computation in networks of passively mobile finite-state sensors. In *Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC04)*, pages 290–299. ACM.

Angluin, D., Aspnes, J., and Eisenstat, D. (2006). Stably computable predicates are semilinear. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC06)*, pages 292–299. ACM.

Apt, K. and Kozen, D. C. (1986). Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22(6):307–309.

Baier, C. and Katoen, J. P. (2008). *Principles of Model Checking (Representation and Mind Series)*. The MIT Press.

Belardinelli, F., Grossi, D., and Lomuscio, A. (2015). Finite abstractions for the verification of epistemic properties in open multi-agent systems. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI15)*, pages 854–860. AAAI Press.

Belardinelli, F., Kouvaros, P., and Lomuscio, A. (2017). Parameterised verification of data-aware multi-agent systems. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI17)*, pages 98–104. AAAI Press.

Bertrand, N. and Fournier, P. (2013). Parameterized verification of many identical probabilistic timed processes. In *Proceedings of the 33rd Foundations of Software Technology and Theoretical Computer Science conference (FSTTCS13)*, pages 501–513. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Bianco, A. and de Alfaro, L. (1995). Model checking of probabalistic and nondeterministic systems. In *Proceedings of the 15th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS95)*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer.

Bloem, R., Jacobs, S., Khalimov, A., Konnov, I., Rubin, S., Veith, H., and Widder, J. (2015). *Decidability of Parameterized Verification*. Morgan and Claypool Publishers.

Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm intelligence*. Oxford University Press.

Bozzano, M. and Villafiorita, A. (2007). The FSAP/NuSMV-SA safety analysis platform. *Software Tools for Technology Transfer*, 9(1):5–24.

Campo, A. and Dorigo, M. (2007). Efficient multi-foraging in swarm robotics. In *Advances in Artificial Life*, pages 696–705. Springer.

Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., and Simaitis, A. (2013). Automatic

verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92.

Chen, T. and Lu, J. (2007). Probabilistic alternating-time temporal logic and model checking algorithm. In *Proceedings of the 4th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD07)*, pages 35–39. IEEE Computer Society.

Clarke, E., Grumberg, O., and Browne, M. (1989). Reasoning about networks with many identical finite state processes. *Information and Computation*, 81(1):13–31.

Clarke, E. M., Emerson, E. A., and Sistla, A. P. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263.

Clarke, E. M., Grumberg, O., and Peled, D. A. (1999). *Model Checking*. The MIT Press, Cambridge, Massachusetts.

Courcoubetis, C. and Yannakakis, M. (1995). The complexity of probabilistic verification. *J. ACM*, 42(4):857–907.

de Oca, M., Ferrante, E., Scheidler, A., Pinciroli, C., Birattari, M., and Dorigo, M. (2011). Majority-rule opinion dynamics with differential latency: a mechanism for self-organized collective decision-making. *Swarm Intelligence*, 5(3-4):305–327.

Dehnert, C., Junges, S., Katoen, J., and Volk, M. (2017). A storm is coming: A modern probabilistic model checker. In *Proceedings of the 29th International Conference on Computer Aided Verification (CAV17)*, volume 10427 of *Lecture Notes in Computer Science*, pages 592–600. Springer.

Dixon, C., Winfield, A., Fisher, M., and Zeng, C. (2012). Towards temporal verification of swarm robotic systems. *Robotics and Autonomous Systems*, 60(11):1429–1441.

Duflot, M., Kwiatkowska, M., Norman, G., Parker, D., Peyronnet, S., Picaronny, C., and Sproston, J. (2010). *FMICS Handbook on Industrial Critical Systems*, chapter Practical Applications of Probabilistic Model Checking to Communication Protocols, pages 133–150. IEEE Computer Society Press.

Esparza, J., Ganty, P., Leroux, J., and Majumdar, R. (2017). Verification of population protocols. *Acta Informatica*, 54(2):191–215.

Ezekiel, J. and Lomuscio, A. (2009). Combining fault injection and model checking to verify fault tolerance in multi-agent systems. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS09)*, pages 113–120. IFAAMAS Press.

Ezekiel, J. and Lomuscio, A. (2017). Combining fault injection and model checking to verify fault tolerance, recoverability, and diagnosability in multi-agent systems. *Information and Computation*, 254(2):167–194.

Ezekiel, J., Lomuscio, A., Molnar, L., and Veres, S. (2011). Verifying fault tolerance and self-diagnosability of an autonomous underwater vehicle. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI11)*, pages 1659–1664. AAAI Press.

Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. Y. (1995). *Reasoning about Knowledge*. MIT Press, Cambridge.

Feller, W. (1968). *An Introduction to Probability Theory and Its Applications*. Wiley.

Ferrante, E., Brambilla, M., Birattari, M., and Dorigo, M. (2013). *Socially-Mediated Negotiation for Obstacle Avoidance in Collective Transport*, volume 83 of *Springer Tracts in Advanced Robotics (STAR)*, pages 571–583. Springer Berlin Heidelberg.

Forejt, V., Kwiatkowska, M., Norman, G., and Parker, D. (2011). Automated verification techniques for probabilistic systems. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, volume 6659 of *Lecture Notes in Computer Science*, pages 53–113. Springer.

Fournier, P. (2015). *Parameterized verification of networks of many identical processes*. PhD thesis, Université de Rennes 1.

Gainer, P., Dixon, C., and Hustadt, U. (2016). Probabilistic model checking of ant-based positionless swarming. In *Proceedings of the 17th Annual Conference Towards Autonomous Robotics (TAROS16)*, pages 127–138. Springer.

Gainer, P., Hahn, E. M., and Schewe, S. (2018). Incremental verification of parametric and reconfigurable markov chains. In *Proceedings of the 15th International Conference on Quantitative Evaluation of Systems (QEST18)*, volume 11024 of *Lecture Notes in Computer Science*, pages 140–156. Springer.

Ginsburg, S. and Spanier, E. (1966). Semigroups, presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296.

Graham, D. (2008). *Parameterised verification of randomised distributed systems using state-based models*. PhD thesis, University of Glasgow.

Graham, D., Calder, M., and Miller, A. (2009). An inductive technique for parameterised model checking of degenerative distributed randomised protocols. *Electronic Notes in Theoretical Computer Science*, 250(1):87–103.

Hansson, H. and Jonsson, B. (1994). A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535.

Huang, X. and Luo, C. (2013). A logic of probabilistic knowledge and strategy. In *Proceedings*

of the 12th International Conference on Autonomous Agents and Multi-Agent systems (AAMAS13), pages 845–852. IFAAMAS.

Jacobsen, L., Jacobsen, M., Møller, M., and Srba, J. (2011). Verification of timed-arc petri nets. In *Proceedings of the 37th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM11)*, volume 6543 of *Lecture Notes in Computer Science*, pages 46–72. Springer.

John, A., Konnov, I., Schmid, U., Veith, H., and Widder, J. (2013). Parameterized model checking of fault-tolerant distributed algorithms by abstraction. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 201–209. IEEE.

Kaiser, A., Kroening, D., and Wahl, T. (2010). Dynamic cutoff detection in parameterized concurrent programs. In *Proceedings of the 22nd International Conference on Computer Aided Verification (CAV10)*, volume 6184 of *Lecture Notes in Computer Science*, pages 645–659. Springer.

Katoen, J.-P. (2016). The probabilistic model checking landscape. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 31–45. ACM.

Kemeny, J. G., Snell, J. L., and Knapp, A. W. (1976). *Denumerable Markov Chains*. Graduate Texts in Mathematics. Springer.

Konur, S., Dixon, C., and Fisher, M. (2012). Analysing robot swarm behaviour via probabilistic model checking. *Robotics and Autonomous Systems*, 60(2):199–213.

Kouvaros, P. (2015). *Parameterised Verification for Multi-Agent Systems*. PhD thesis, Imperial College London.

Kouvaros, P. and Lomuscio, A. (2013). A cutoff technique for the verification of parameterised interpreted systems with parameterised environments. In *Proceedings of the 23rd Inter-*

national Joint Conference on Artificial Intelligence (IJCAI13), pages 2013–2019. AAAI Press.

Kouvaros, P. and Lomuscio, A. (2015). Verifying emergent properties of swarms. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI15)*, pages 1083–1089. AAAI Press.

Kouvaros, P. and Lomuscio, A. (2016). Parameterised verification for multi-agent systems. *Artificial Intelligence*, 234:152–189.

Kouvaros, P. and Lomuscio, A. (2017a). Parameterised verification of infinite state multi-agent systems via predicate abstraction. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI17)*, pages 3013–3020. AAAI Press.

Kouvaros, P. and Lomuscio, A. (2017b). Verifying fault-tolerance in parameterised multi-agent systems. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI17)*, pages 288–294. AAAI Press.

Kouvaros, P., Lomuscio, A., and Pirovano, E. (2018). Symbolic synthesis of fault-tolerance ratios in parameterised multi-agent systems. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence and 23rd European Conference on Artificial Intelligence (IJCAI-ECAI18)*, pages 324–330. IJCAI.

Kouvaros, P., Lomuscio, A., Pirovano, E., and Punchihewa, H. (2019). Formal verification of open multi-agent systems. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS19)*, pages 179–187. ACM.

Kwiatkowska, M., Norman, G., and Parker, D. (2007). Stochastic model checking. In *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *Lecture Notes in Computer Science*, pages 220–270. Springer.

Kwiatkowska, M., Norman, G., and Parker, D. (2010). *Symbolic Systems Biology*, chapter Probabilistic Model Checking for Systems Biology, pages 31–59. Jones and Bartlett.

Kwiatkowska, M., Norman, G., and Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV11)*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer.

Kwiatkowska, M., Norman, G., Parker, D., and Santos, G. (2018a). Automated verification of concurrent stochastic games. In *Proceedings of the 15th International Conference on Quantitative Evaluation of SysTems (QEST18)*, volume 11024 of *Lecture Notes in Computer Science*, pages 223–239. Springer.

Kwiatkowska, M., Parker, D., and Wiltsche, C. (2018b). PRISM-games: verification and strategy synthesis for stochastic multi-player games with multiple objectives. *Software Tools for Technology Transfer*, 20(2):195–210.

Kwiatkowska, M. Z., Norman, G., and Parker, D. (2012). The PRISM benchmark suite. In *Proceedings of the 9th International Conference on Quantitative Evaluation of Systems (QEST12)*, pages 203–204. IEEE Computer Society.

Lerman, K. and Galstyan, A. (2002). Mathematical model of foraging in a group of robots. *Autonomous Robots*, 13(2):127–141.

Leroux, J. and Schmitz, S. (2015). Demystifying reachability in vector addition systems. In *Proceedings of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS15)*, pages 56–67. IEEE.

Liu, W. and Winfield, A. (2010). Modeling and optimization of adaptive foraging in swarm robotic systems. *International Journal of Robotics Research*, 29(14):1743–1760.

Lomuscio, A. and Michaliszyn, J. (2015). Verifying multi-agent systems by model checking three-valued abstractions. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS15)*, pages 189–198. ACM.

Lomuscio, A., Penczek, W., and Qu, H. (2010). Partial order reduction for model checking interleaved multi-agent systems. *Fundamenta Informaticae*, 101(1–2):71–90.

Lomuscio, A. and Pirovano, E. (2018). Verifying emergence of bounded time properties in probabilistic swarm systems. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI-ECAI18)*, pages 403–409. IJCAI.

Lomuscio, A. and Pirovano, E. (2019). A counter abstraction technique for the verification of probabilistic swarm systems. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS19)*, pages 161–169. ACM.

Lomuscio, A. and Pirovano, E. (2020a). Parameterised verification of strategic properties in probabilistic multi-agent systems. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS20)*, pages 762–770. ACM.

Lomuscio, A. and Pirovano, E. (2020b). Verifying fault-tolerance in probabilistic swarm systems. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI-PRICAI2020)*, pages 325–331. IJCAI.

Lomuscio, A., Qu, H., and Raimondi, F. (2017). MCMAS: A model checker for the verification of multi-agent systems. *Software Tools for Technology Transfer*, 19(1):9–30.

Lukina, A., Tiwari, A., Smolka, S. A., and Grosu, R. (2018). Adaptive neighborhood resizing for stochastic reachability in multi-agent systems. *CoRR*, abs/1805.07929.

Maggi, F. M., Montali, M., and Peñaloza, R. (2019). Probabilistic temporal logic over finite traces. *CoRR*, abs/1903.04940.

MCMAS-P (2015). Model Checking Parameterised Multi-Agent Systems http://vas.doc.ic.ac.uk/software/extensions.

Meyer, R. (2009). A theory of structural stationarity in the π-calculus. *Acta Informatica*, 46(2):87–137.

Moses, Y. and Zamir, N. (2020). Probably approximately knowing. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC20)*, pages 375–384. ACM.

Nembrini, J. (2005). *Minimalist Coherent Swarming of Wireless Networked Autonomous Mobile Robots*. PhD thesis, University of the West of England.

Norman, G., Parker, D., and Sproston, J. (2013). Model checking for probabilistic timed automata. *Formal Methods in System Design*, 43(2):164–190.

Parrott, C., Dodd, T., Boxall, J., and Horoshenkov, K. (2020). Simulation of the behavior of biologically-inspired swarm robots for the autonomous inspection of buried pipes. *Tunnelling and Underground Space Technology*, 101:103356.

Penczek, W. and Pólrola, A. (2004). Specification and model checking of temporal properties in time Petri nets and timed automata. In *Proceedings of the 25th International Conference on Applications and Theory of Petri Nets (ATPN04)*, volume 3099 of *Lecture Notes in Computer Science*, pages 37–76. Springer.

Pnueli, A. (1977). The temporal logic of programs. In *Proceedings of the 18th International Symposium Foundations of Computer Science (FOCS77)*, pages 46–57. IEEE.

Pnueli, A., Xu, J., and Zuck, L. (2002). Liveness with (0, 1,infinity)-counter abstraction. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 93–111. Springer.

Puterman, M. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.

Rackoff, C. (1978). The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231.

Reisig, W. (1985). *Petri Nets. An Introduction*, volume 4 of *EACTS Monographs on Theoretical Computer Science*. Springer.

Şahin, E. (2005). Swarm robotics: From sources of inspiration to domains of application. In *Proceedings of the 2004 International Conference on Swarm Robotics (SAB04)*, volume 3342 of *Lecture Notes in Computer Science*, pages 10–20. Springer.

Şahin, E. and Winfield, A. (2008). Special issue on swarm robotics. *Swarm Intelligence*, 2(2):69–72.

Spalazzi, L. and Spegni, F. (2020). Parameterized model checking of networks of timed automata with boolean guards. *Theoretical Computer Science*, 813:248–269.

Vardi, M. Y. (1985). Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FOCS85)*, pages 327–338. IEEE.

Winfield, A., Liu, W., Nembrini, J., and Martinoli, A. (2008). Modelling a wireless connected swarm of mobile robots. *Swarm Intelligence*, 2(2-4):241–266.

Winfield, A., Sa, J., Fernández-Gago, M., Dixon, C., and Fisher, M. (2005). On formal specification of emergent behaviours in swarm robotic systems. *International Journal of Advanced Robotic Systems*, 2(4):363–370.

Zhu, Q., Li, H., Han, Z., and Basar, T. (2010). A stochastic game model for jamming in multi-channel cognitive radio systems. In *Proceedings of IEEE International Conference on Communications (ICC10)*, pages 1–6. IEEE.