Department of Computing
Imperial College London

Parameterised Verification for Multi-Agent Systems

Panagiotis Kouvaros

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of Imperial College London
and the Diploma of Imperial College London, October 2015

Abstract

In the past ten years several methods have been put forward for the efficient model checking of multiagent systems against agent-based specifications. Yet, since the number of states is exponential in the number of agents in the system, the model checking problem remains intractable for systems of many agents. This is particularly problematic when wishing to reason about unbounded systems where the number of components is not known at design time. Systems ranging from robotic swarms to e-commerce applications constitute typical examples in which the number of participants is independent of the design process.

This thesis develops parameterised model checking techniques for the validation of multiagent systems irrespectively of the number of the agents present. To do this, a semantics that captures parameterised, synchronous multiagent systems and one that models parameterised, interleaved multiagent systems are introduced. Both semantics extend interpreted systems in a parameterised setting where the number of agents is the parameter.

Parameterised model checking techniques for the semantical classes introduced are developed. A sound and complete cutoff methodology is studied for parameterised interpreted systems. A sound but incomplete cutoff procedure for parameterised interleaved interpreted systems is also studied. While the latter procedure is in exponential space, three notable subclasses are isolated and more effective verification techniques are put forward. The algorithms proposed are shown to be sound. For one class the decidability of the verification problem is shown and a complete cutoff procedure is discussed.

Finally, the model checker `MCMAS-P` is introduced. The tool supports the verification of unbounded multiagent systems against temporal-epistemic specifications. `MCMAS-P` implements the procedures here developed; the procedure invoked depends on the properties of the system under examination. Experimental results obtained on cache coherence protocols, mutual exclusion protocols, swarm foraging algorithms, and swarm aggregation algorithms are reported.

To the memory of Michalis Kouvaros

Acknowledgements

I would like to acknowledge my supervisor Prof. Alessio Lomuscio for his guidance and support.

I would also like to thank my MSc supervisor Prof. Ian Hodkinson, who taught me logic, and whose excitement and knowledge inspired me to pursue a doctorate degree.

I am grateful to the Department of Computing of Imperial College London for giving me the opportunity to carry out research as a doctoral student in computer science. I acknowledge the financial support of the EPSRC Research Project Trusted Autonomous Systems (EP/I00520X/1).

Last but not least, I would like to thank my family for their unconditional love, support, and encouragement. This thesis is dedicated to the memory of my brother Michalis Kouvaros.

Declaration of Originality

I hereby declare that the research composing this thesis is my own, except where otherwise acknowledged.

Copyright Declaration

Contents

List of Tables

List of Algorithms

List of Figures

Chapter 1

Introduction

Multiagent systems (MAS) are distributed computing systems where every component, or agent, is autonomous in its actions, adaptive to its environment, and proactive in achieving its goals [Wei99, SLB08, Woo09]. With the development and deployment of MAS in diverse applications, such as search-and-rescue [Mur00], web-services [MS04], personal negotiation assistants [RZSH+14], there is an increasing need to study powerful and versatile methodologies for their verification [LQR15, GvdM04, KNN+08, CLMM15, WLP05, BFVW06, HW02a, LPQ10, KP04b, DWFZ12]. A key technique that has emerged in the past ten year is that of model checking [CGP99, HR00]. In model checking a system is represented by a semantical structure, or a model, and a specification is encoded as a logic formula. Then, an automatic procedure establishes whether the model satisfies the formula.

Differently from reactive systems, where plain temporal formulae are used, specifications for MAS are given in richer logics that represent high-level attitudes of agency. In particular, model checking MAS against epistemic specifications has proven useful in applications such as autonomous vehicles [ELMV11], service-oriented computing [LQS11] and security [BCL09].

Model checking techniques are often based on exhaustive search of the state space of the model, thereby providing full guarantees of correctness. However, the size of the state space grows exponentially in the number of variables encoding the system. As a result, MAS with many components remain hard to model check.

Nevertheless, a number of methodologies have been put forward to overcome the state explosion problem. Techniques such as symbolic model checking [LQR15], abstraction [CDLR09], partial order reduction [LPQ10], bounded model checking [WLP05], have made it possible to verify systems with very large state spaces (see section 2.4 for a further discussion).

Although these are significant results, a fundamental assumption in most of these techniques is that all the components present in the MAS are defined and modelled at design time. But what if the MAS is deployed in varying number of components depending on the domain? We refer to these systems as *unbounded MAS* (UMAS). UMAS are MAS where the number of components is not known at design time. Insect-size surveillance vehicles, search and rescue missions, industrial supply chains, scenarios in the Internet of Things, multi-party negotiation protocols and auctions, voting protocols, e-services, are all examples of UMAS. In these cases, one could encode a system with a given number of agents and verify that a specification holds. However, additional agents may possibly interfere with the system in unpredictable ways resulting in the specifications being violated. Therefore, to fully verify the system, the process would have to be repeated for any possible number of components. This is computationally impossible.

The above verification problem cannot be solved by a trivial application of standard model checking as this would entail model checking an infinite number of systems. To solve this, we require a technique that can be applied independently of the number of agents in the system.

The overall objective of this thesis is the development of automatic methodologies for the verification of unbounded multiagent systems irrespectively of the number of agents present.

1.1 Objectives

We break down the above overall objective into three objectives. We begin with a short discussion on *parameterised model checking* (for a further discussion, see section 2.5).

Parameterised model checking is a model checking approach that is concerned with the decision problem of establishing whether a certain specification holds for a system composed of *any* number of *homogeneous* participants. The approach is extensively studied in the context of reactive systems [EK00, AKR+14, ACJT96, CTTV04, ADHR07, CGB89, EN98, EFM99, EK03a, KKW10, PS00, ZP04, CTV06, APR+01, EN96, HBR09, GS92, AHH13, DSZ10, EN95]. These lines of work formulate the parameterised model checking problem in a finitary way by giving a *behavioural template* for the participants and the specification to be checked. Given the parameter for the actual number of participants in the system, the concrete system of these participants can be build. The standard model checking problem can then be solved for the concrete system.

The first objective of this thesis is the formulation of the parameterised model checking problem for UMAS.

Differently from the existing literature, the above objective targets AI-based, autonomous MAS [Woo09]. These require different primitives from reactive systems both in terms of modelling and specifications. In semantic terms, we are inspired by the longstanding tradition in MAS to model systems of agents in terms of interpreted systems [FHMV95]. To account for an unbounded number of agents, we introduce abstract *agent templates* from which concrete interpreted systems can be generated. Emphasis is given to the synchronisation patterns that may emerge between the agents and the environment, and not to particular topologies as in much of the existing literature [EN95, DSZ10, DSZ11, CTTV04, AJKR14, EK04, HSBR10, HBR09]. Additionally, emphasis is given to rich temporal-epistemic formulae to express specifications, and not just safety and liveliness properties, as in much of the existing literature [Mai01, BLS02, KKW10, PXZ02, ACJT96, EN98, EFM99].

As said above, a naive approach to solving the parameterised model checking problem entails checking an infinite number of systems, i.e., all possible systems built from any number of agents. Given that the number of agents is not bounded, it also entails checking systems of unbounded size. Nevertheless, some approaches to the problem are motivated by strong empirical evidence that unbounded systems often exhibit a *cutoff* [EK00, KKW10, AHH13, EN95, HSBR10, HBR09, AJKR14, AKR+14]. A cutoff expresses the number of components that is sufficient to analyse when evaluating a given specification. That is, given a cutoff, the parameterised model checking problem is reduced to the problem of model checking each concrete system up to the cutoff. This leads to the following.

The second objective of this thesis is the development of sound, automatic cutoff identification procedures for UMAS.

We now discuss the third and final objective. Along with theoretical investigations, a number of automatic tools were developed for the verification of MAS. Most of these are based on symbolic data structures. Most of these support epistemic specifications [LPQ10, LPW07, MS99, PL03, vdMS04, RL05]. Others target deontic specifications [RL04, WLP05, CLMM14, CLMM15] or specifications expressing strategic abilities [KP04a, LR06b]. Symbolic checkers such as MCK [GvdM04], MCMAS [LQR09] and VERICS [KNN+08] are all capable of handling state spaces in the region of 10^{15} and beyond. However, they do not support parameterised verification.

The third objective of this thesis is to extend the functionality of agent-based model checking tools to support parameterised verification for UMAS.

To summarise, the thesis's goal is to put forward techniques to solve the parameterised model checking problem for UMAS. To do this, the parameterised model checking problem will be formulated in the context of multiagent systems. Following this, cutoff methodologies will be developed to solve the problem. Finally, a tool realising these techniques will be presented.

1.2 Contributions

The main contributions of this thesis are categorised and listed as follows:

1. Conceptual

    - We introduce *parameterised interpreted systems* (PIS), a semantics for synchronous UMAS.
    - We introduce *parameterised interleaved interpreted systems* (PIIS), a semantics for asynchronous UMAS.
    - We introduce *indexed ACTL*K*, an indexed temporal-epistemic logic for expressing properties of UMAS.

- We introduce the notion of *cycle-stuttering simulation*. This can be used to show behavioural equivalence between two systems.

2. Methodological

   - We introduce an automatic procedure for the verification of PIS.
   - We introduce an automatic procedure for the verification of PIIS.
   - We introduce three automatic procedures for the verification of three subclasses of PIIS.

3. Theoretical

   - We prove the verification procedure for PIS to be sound and complete.
   - We prove the verification procedure for PIIS to be sound.
   - For each of the subclasses of PIIS, we prove the associated verification procedure to be sound. For one class we show a sound and complete result.

4. Implementation

   - We introduce `MCMAS-P`, a tool for model checking PIIS.

1.3 Research output

The majority of the results composing this thesis were presented, in shorter form, in the following papers:

- P. Kouvaros, A. Lomuscio, "Parameterised Verification for Multiagent Systems", Artificial Intelligence, 2016.
- P. Kouvaros, A. Lomuscio, "Verifying Emergent Properties of Swarms", Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI15), 2015.
- P. Kouvaros, A. Lomuscio, "A Counter Abstraction Technique for the Verification of Robot Swarms", Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI15), 2015.

- P. Kouvaros, A. Lomuscio, "A Cutoff Technique for the Verification of Parameterised Interpreted Systems with Parameterised Environments", Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI13), 2013.
- P. Kouvaros, A. Lomuscio, "Automatic Verification of Parameterised Interleaved Multi-Agent Systems", Proceedings of the 12th International Conference on Autonomous Agents and Multi-Agent systems (AAMAS13), 2013.

1.4 Summary of contents

The rest of the thesis is organised as follows.

Chapter 2 introduces the technical background to this thesis.

In chapter 3 we introduce PIS and PIIS. In particular, we give the definitions for the agent templates, the environment template, and we identify different types of interactions that the agents and the environment can engage in. This is followed by the introduction of the specification language which consists of an indexed version of a temporal-epistemic logic. The chapter continues with some key theoretical observations regarding the notions of stuttering simulations and cycle-stuttering simulations that can be defined on these systems and an exploration on the extent to which these preserve logical satisfaction.

Chapters 5, 6 and 7 report the main theoretical results of the thesis. In chapter 4 we study parameterised interpreted systems, and we give a sound and complete parameterised model checking procedure. In chapter 5 we investigate parameterised interleaved interpreted systems, and we present a sound but incomplete procedure. In chapter 6 we define three special classes of PIIS, each representing different synchronisation patterns for the agents in the system. For each of these classes we introduce a sound parameterised model checking procedure. For one class we provide a sound and complete procedure.

Chapter 7 reports an implementation that we built from the techniques described in Chapters 5 and 6. Specifically, the chapter introduces `MCMAS-P`, an extension of `MCMAS`, a model checker for the verification of MAS. As we will explain, `MCMAS-P` conducts an iterative check on the existence of certain simulations that guarantee, by the methods of chapters 5 and 6, that a cutoff exists. If this can be shown, the checker performs plain model checking on corresponding concrete systems in line with the requirements of the theory developed. The chapter reports the experimental results obtained on the Alpha swarm aggregation protocol [WLNM08], on a swarm foraging algorithm [Liu07], on the MSI, MESI, and MOESI cache coherence protocols, and on the Train-Gate-Controller [HW02b, LPQ10].

We conclude in chapter 8 where we summarise the contributions of the thesis; we compare them with related work; and we identify possible future work.

Chapter 2

Background

2.1 Interpreted systems

Interpreted systems are a standard semantics for describing multi-agent systems [FHMV95]. They provide a natural setup to interpret specifications in a variety of languages including temporal-epistemic logic and alternating temporal logic [FHMV95, AHK02]. The formalism typically assumes synchrony between the agents, i.e, the agents evolve *simultaneously* by performing an action at every tick of an external global clock. In many settings, however, e.g., games [FHMV95], swarm robotics [BDT99], agents operate in terms of internal clocks and can evolve in an interleaved, asynchronous fashion. Interleaved interpreted systems are a class of interpreted systems constraining the interleaved evolution of the agents' actions [LPQ10]. This section summarises interpreted systems and interleaved interpreted systems.

2.1.1 Interpreted systems

We begin with a MAS composed of n agents $\mathcal{A} = \{1, \ldots, n\}$ acting in an environment E. The environment is treated as a special agent allowing us to consider a MAS as composed of the set $\mathcal{A} \cup \{E\}$ of agents. Each agent $i \in \mathcal{A} \cup \{E\}$ is described by the following.

- A nonempty and finite set of *local states* L_i. A local state encodes all the information accessible to the agent in the current configuration of the system.

- A nonempty and finite set of *local actions* Act_i. Local actions represent the possible actions that the agent can perform.
- A *protocol* P_i that governs which actions may be performed at a given local state. The protocol is a function $P_i : L_i \to \mathcal{P}(Act_i)$ that assigns a set of actions to each local state.
- An *evolution function* t_i that determines the temporal evolution of the agent's local states. The evolution function is a function $t_i : L_i \times Act_1 \times \ldots \times Act_n \times Act_E \to L_i$ that returns the next local state given the agent's current local state, its action, and all other agents' actions.

Local states are private in the sense that they can not be observed by other agents, whereas local actions are global since they can be "seen" by other agents. Whenever more than one action is enabled for an agent at a given local state, the agent is assumed to select non-deterministically which action to perform.

A *global state* $g = (l_1, \ldots, l_n, l_E)$ is a tuple of local states for all the agents in the system; it describes the system at a particular instant of time. Given a global state $g = (l_1, \ldots, l_n, l_E)$ and an agent i, we write $ls_i(g)$ to denote the local state $ls_i(g) = l_i$ of agent i in g. The set S of all possible global states is the Cartesian product $S = L_1 \times \ldots \times L_n \times L_E$ of the agents' sets of local states. A *joint action* $\overline{a} = (a_1, \ldots, l_n, l_E)$ is a tuple of local actions for all the agents in the system. For a joint action $\overline{a} = (a_1, \ldots, l_n, l_E)$ and an agent i, we denote the local action of agent i in $\overline{a}$ by $la_i(\overline{a})$. The set $ACT = Act_1 \times \ldots Act_n \times Act_E$ of all possible joint actions is the Cartesian product of the agents' sets of local actions.

We now have all the necessary ingredients to define an interpreted system.

Definition 2.1 (Interpreted system). *Let AP be a set of atomic propositions. An* interpreted system *IS is a tuple*

$$IS = \big((L_i, Act_i, P_i, t_i)_{i \in \mathcal{A}}, (L_E, Act_E, P_E, t_E), I, V\big)$$

where $I \subseteq S$ is a set of initial global states and $V : S \to \mathcal{P}(AP)$ is a valuation function on the set S of possible global states.

The system's global states evolve over time in compliance with the agents' local protocols and local evolution functions, thereby inducing a global transition function.

Definition 2.2 (Global transition function)**.** *The* global transition function $t : S \times ACT \to S$ *is defined as follows.*

$$t(g, \overline{a}) = g' \textit{ iff } t_i(ls_i(g), la_i(\overline{a})) = ls_i(g') \textit{ and } la_i(\overline{a}) \in P_i(ls_i(g)) \textit{ for all } i \in \mathcal{A} \cup \{E\}$$

We sometimes write the above as $g \to_{\overline{a}} g'$.

A path π is a finite or infinite sequence $\pi = g^1 \overline{a}^1 g^2 \overline{a}^2 g^3 \ldots$ with $g^i \to_{\overline{a}^i} g^{i+1}$, for every $i \geq 1$ whenever applicable. Given a path π, we write $\pi(i)$ for the i-th state in π. By $\pi[i]$, we denote the suffix $g^i a^i g^{i+1} \ldots$ of π. The set of all paths originating from a state g is denoted by $\Pi(g)$. A global state g is said to be reachable from a global state g^1 if there is a path $\pi \in \Pi(g^1)$ such that $\pi(i) = g$ for some $i \geq 1$.

Given an interpreted system IS, we can associate a Kripke model $\mathcal{M}_{IS}$ to IS that can be used to interpret temporal-epistemic formulae [FHMV95].

Definition 2.3 (Model)**.** *Given an interpreted system IS, its associated Kripke model $\mathcal{M}_{IS}$ is a tuple $\mathcal{M}_{IS} = (G, I, R, (\mathcal{K}_i)_{i \in \mathcal{A}}, V)$, where*

- *$G \subseteq S$ is the set of* reachable global states*, i.e, the set of states reachable via the global transition function from the set of initial global states.*
- *$R \subseteq G \times G$ is the global transition relation defined as $(g, g') \in R$ iff there is a joint action $\overline{a}$ such that $g \to_{\overline{a}} g'$.*
- *$\mathcal{K}_i \subseteq G \times G$ is the epistemic accessibility relation for agent i defined on local equalities for the agents' states: $\mathcal{K}_i(g, g')$ iff $ls_i(g) = ls_i(g')$.*

See section 2.2 for the definition of the satisfaction relation $\models$ for CTL*K.

2.1.2 Interleaved interpreted systems

We have thus far considered actions in interpreted systems to be performed at the same round by all the agents. This gives a natural way to model synchronous multiagent systems. To model asynchronous multiagent systems, interleaved interpreted systems have been put forward [LPQ10]. Interleaved interpreted systems insist on only one local action to be performed at a given time. It is further assumed that if more than one agents admit in their repertoire of actions the action to be performed, then all agents sharing [1] the action have to perform it at the round. Additionally, interleaved interpreted systems deviate from the standard treatment of interpreted systems by assuming the resulting local state of an agent to depend only on the agent's action at the previous round as opposed to all the agents' actions. The agents are therefore communicating by means of shared actions. We below make this discussion precise by giving the formal definition of an interleaved interpreted system.

Let us assume a MAS composed of n agents $\mathcal{A} = \{1, \ldots, n\}$ operating in an environment E. The description of an agent $i \in \mathcal{A} \cup \{E\}$ includes a nonempty and finite set of local states L_i, a unique inital state ι_i, and a nonempty and finite set of actions Act_i. Actions are performed in compliance with a protocol $P_i : L_i \to Act_i$ that selects which actions can be performed at a given local state. The evolution of an agent i's local states is specified by a transition function $t_i : L_i \times Act_i \to L_i$ returning the next local state given the agent's (current) local state and action. Note that the local transition function considered here has the local action as the only parameter.

A "null" action ϵ_i is assumed to be a member of any set Act_i. It is assumed that for every state $l_i \in L_i$ we have that: (i) $\epsilon_i \in P_i(l_i)$ (i.e., the null action is enabled at every local state); (ii) $t_i(l_i, \epsilon_i) = l_i$ (i.e., an agent stutters in its current local state whenever it performs the null action).

We now define interleaved interpreted systems.

Definition 2.4 (Interleaved interpreted system). *Let AP be a set of atomic propositions. An*

[1] A local action is said to be *shared* by two or more agents if said agents admit the action in their repertoire of actions.

interleaved interpreted system is a tuple

$$IIS = \big((L_i, \iota_i, Act_i, P_i, t_i)_{i \in \mathcal{A}}, (L_E, \iota_E, Act_E, P_E, t_E), \iota, V\big)$$

where $\iota = (\iota_1, \ldots, \iota_n, \iota_E)$ is the unique initial global state, and $V : S \to \mathcal{P}(AP)$ is a valuation function on the set S of possible global states.

To define the global transition function, given an action $a \in \bigcup_{i \in \mathcal{A} \cup \{E\}} Act_i$, let $Agent(a) = \{i \in \mathcal{A} \cup \{E\} : a \in Act_i\}$ to be the set of agents admitting the action in their repertoire.

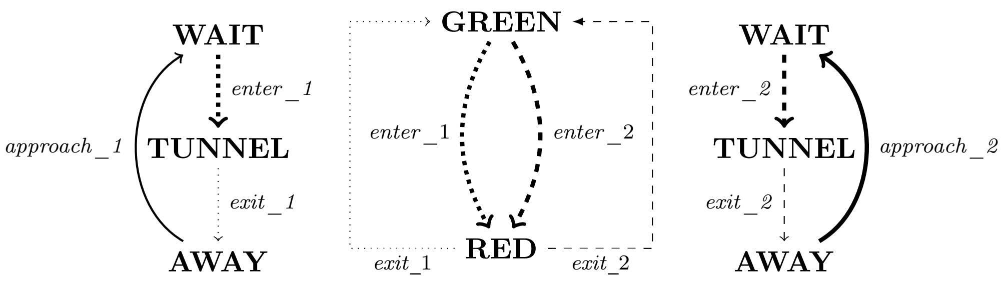
Definition 2.5 (Global transition function)**.** *The global transition function $t : S \times ACT \to S$ is defined as follows: $t(g, \overline{a}) = g'$ iff there is an action $a \in Act_1 \cup \ldots \cup Act_n \cup Act_E$ such that:*

- *for all $i \in Agent(a)$, we have that $la_i(\overline{a}) = a$, $la_i(\overline{a}) \in P_i(ls_i(g))$, and $t_i(ls_i(g), la_i(\overline{a})) = ls_i(g')$;*
- *and for all $i \in (\mathcal{A} \cup \{E\}) \setminus Agent(a)$, we have that $la_i(\overline{a}) = \epsilon_i$ and $t_i(ls_i(g), la_i(\overline{a})) = ls_i(g') = ls_i(g)$.*

In short we write the above as $g \to_a g'$.

In other words, at each round, all agents participating in the global transition are required to perform the same local action. The agents not participating in the global transition are assumed to perform the null action. Every agent admitting said local action in its repertoire has to perform it at the round; if there is a local protocol not permitting this, then the local action cannot be performed at the round. So, communication in interleaved interpreted systems is by means of shared actions.

Example 2.1. *Figure 2.1 presents the interleaved interpreted system of the untimed version of the Train-Gate-Controller (TGC) as presented in [HW02b] and adapted from [AdAH$^+$00]. The system of TGC is composed of a controller and two trains. Each train runs along a circular track and both tracks pass through a narrow tunnel. The tunnel can accommodate only one train at any time. Both sides of the tunnel are equipped with traffic lights, which can be either green or red. The controller operates the colour of the traffic lights to let the trains enter and exit the tunnel.*



(a) Train 1. (b) Controller. (c) Train 2.

Figure 2.1: The interleaved interpreted system for the Train-Gate-Controller.

In the figure, the transitions that are depicted with the same style of edges are synchronised. For clarity, null ϵ *actions are omitted in the figure.*

Consider train 1. The train is initially in state $WAIT$ *where it waits to enter the tunnel. The controller, initially in state* $GREEN$*, may synchronise with train 1 on the action* $enter\_1$*. Following this, train 1 goes to state* $TUNNEL$*, whereas the controller goes to state* RED*. As a result, train 2 cannot perform the* $enter\_2$ *action and enter the tunnel, since the action is not enabled by the controller's protocol at state* RED*. Instead, the controller's protocol enables the action* $exit\_1$ *on which action it may synchronise with Train 1. Upon this synchronisation, train 1 goes to state* $AWAY$ *and the controller goes to state* $GREEN$*. Then, train 1 can perform the action* $approach\_1$ *and go back to state* $WAIT$ *where it may again synchronise with the controller and enter the tunnel, or train 2 may synchronise with the controller on the* $enter\_2$ *action and enter the tunnel.*

Given an interleaved interpreted system IIS, a Kripke model $\mathcal{M}_{IIS}$ can be associated to IIS in a similar manner to Definition 2.3. We use the term "model" throughout the thesis to mean either an interpreted system or an interleaved interpreted system, if not otherwise stated.

2.2 Temporal-epistemic logics

Combinations of branching time logic (CTL) [CE81, Eme90, BAPM83] and linear time logic (LTL) [Pnu77, CGP99, HR00] with knowledge have long been used to express properties of agents in a MAS [FHMV95, LR06a, LQR09, KL13a]. This thesis is concerned with the temporal-epistemic logic CTL*K [LPQ10]. CTL*K combines the epistemic logic S5 [2] [FHMV95] with the temporal logic CTL* [HR00], a logic combing the expressive power of LTL and CTL. We recall the basic definitions.

There are two types of formulas in CTL*K: *state formulae* which are interpreted over states, and *path formulae* which are interpreted over paths.

Definition 2.6. *(Syntax of CTL*K) State and path formulae of CTL*K over a set AP of propositions and a set $\mathcal{A}$ of agents are inductively defined as follows.*

S1. *every atomic proposition $p \in AP$ is a state formula;*

S2. *if φ and ψ are state formulas, then $\neg\varphi$, $\varphi \vee \psi$ and $K_i\phi(i \in \mathcal{A})$ are state formulas;*

S3. *if φ is a path formula, then $E(\varphi)$ is a state formula;*

P1. *any state formula φ is also a path formula;*

P2. *if φ and ψ are path formulas, then $\neg\varphi$ and $\varphi \vee \psi$ are path formulas;*

P3. *if φ and ψ are path formulas, then so are $X\varphi$ and $\varphi U \psi$.*

The knowledge modality K_i stands for "agent i knows that"; the path quantifier E is read "there exists a path"; the temporal operators X, U denote the "next" and "until" modalities, respectively.

CTL*K formulae are interpreted on a model $M = (G, I, R, (\mathcal{K}_i)_{i\in\mathcal{A}}, V)$ as standard [CGP99, FHV95]: the temporal modalities are interpreted by means of the global transition relation; the epistemic modalities are interpreted over the epistemic accessibility relations. We write $(M, g) \models \phi$ ($(M, \pi) \models \phi$, respectively) to mean that a state formula (path formula, respectively) is true at a state g (path π, respectively) in M. If M is clear from the context, then we simplify the notation to $g \models \phi$ ($\pi \models \phi$, respectively).

Definition 2.7 (Satisfaction of CTL*K). *Given a model $M = (G, R, (\mathcal{K}_i)_{i\in\mathcal{A}}, I, V)$, the satisfaction relation $\models$ is inductively defined as follows.*

[2] S5 is the normal modal logic KT45 of the class of symmetric, transitive, and Euclidean frames.

S1.	$g \models p$	*iff*	$p \in V(g)$ *for any* $p \in AP$;
S2.	$g \models \neg\varphi$	*iff*	$g \not\models \varphi$;
	$g \models \varphi \vee \psi$	*iff*	$g \models \varphi$ or $g \models \psi$;
	$g \models K_i\phi$	*iff*	$g' \models \phi$ *for every* $g' \in G$ *with* $\mathcal{K}_i(g, g')$;
S3.	$g \models E\varphi$	*iff*	*there exists a path* $\pi \in \Pi(g)$ *such that* $\pi \models \phi$;
P1.	$\pi \models \varphi$	*iff*	$\pi(1) \models \varphi$ *for any state formula* φ;
P2.	$\pi \models \neg\varphi$	*iff*	$\pi \not\models \varphi$;
	$\pi \models \varphi \vee \psi$	*iff*	$\pi \models \varphi$ or $\pi \models \psi$,
P3.	$\pi \models X\phi$	*iff*	$\pi[2] \models \phi$;
	$\pi \models \varphi U \psi$	*iff*	*there is an* $i \geq 1$ *such that* $\pi[i] \models \psi$ *and* $\pi[j] \models \phi$ *for all* $1 \leq j < i$.

A formula ϕ is said to be true in M, denoted $M \models \phi$, if for all $g \in I$, we have that $g \models \phi$. The customary abbreviations of *truth* and *falsity* are assumed: $\top \triangleq p \vee \neg p$, $\bot \triangleq p \wedge \neg p$, for some $p \in AP$. The boolean connective $\wedge$ is given by $\phi \wedge \psi \triangleq \neg(\neg\phi \vee \neg\psi)$. Dual modalities prefixed by the path quantifier A ("for all paths") can be defined as standard [Huth and Ryan, 2004]: $A\phi \triangleq \neg E \neg \phi$. Further, we define $F\phi \triangleq U(\top, \phi)$ with the usual meaning of "Eventually ϕ", and $G\phi \triangleq \neg F \neg \phi$ standing for "Always ϕ".

Some important fragments of CTL*K are defined as follows.

Definition 2.8.

- *LTLK is the fragment of CTL*K in which all formulae are of the form* $A\phi$, *where* ϕ *does not contain the path quantifiers* E *and* A.

- *CTLK is the fragment of CTL*K in which the path quantifiers* E, A *and the temporal modalities* X, U, F, G *may appear only in combinations* $EX, EU, EF, EG, AX, AU, AF, AG$. *That is, CTLK is the logic obtained by removing clauses S3, P1, P2, P3 from Definition 2.6 and adding the following clause:*

  S3'. if ϕ *and* ψ *are state formulas, then* $AX\phi$, $A(\phi U \psi)$ *are state formulas.*

- *ACTL*K is the* universal *fragment of CTL*K in which the path quantifier* E *does not appear* *in any formula, and the negation appears only in subformulas not containing any temporal or epistemic modalities.*

- *ECTL*K is the* existential *fragment of CTL*K in which the path quantifier A does not appear in any formula, and the negation appears only in subformulas not containing any temporal or epistemic modalities.*
- *LTL, CTL, ACTL*, ECTL* are fragments of LTLK, CTLK, ACTL*K, ECTL*K, respectively, in which epistemic modality K does not appear in any formula.*
- *For any logic L, the* stuttering-insensitive *fragment of L, denoted $L\backslash X$, is the fragment of L in which the temporal modality X does not appear in any formula.*

2.3 Equivalences

Simulations are certain relations between models that preserve the satisfaction of modal formulas [Mil80]. Several notions of simulations attracted much attention in the analysis of model checking techniques [CDLR09, BCG88, CGL94]. The key application of these studies is the simplification of the original model to check to a possibly much smaller *abstract* model by identifying simulation relations between the two models. We define the standard notions of simulations for the temporal-epistemic logics ACTL*K, CTL*K, and their stuttering-insensitive fragments ACTL*K$\backslash X$, CTL*K$\backslash X$, respectively.

2.3.1 Equivalences preserving ACTL*K and CTL*K

A model $\mathcal{M}' = (G', I', R', (\mathcal{K}_i')_{i\in\mathcal{A}}, V')$ simulates a model $\mathcal{M} = (G, I, R, (\mathcal{K}_i)_{i\in\mathcal{A}}, V)$ if every behaviour exhibited by the latter is also exhibited by the former. Since an ACTL*K formula may only quantify over all paths, every ACTL*K formula satisfied by $\mathcal{M}'$ is also satisfied by $\mathcal{M}$.

Definition 2.9 (Simulation [CDLR09]). *A relation $\sim_s \subseteq G \times G'$ is a simulation between two models $\mathcal{M}$ and $\mathcal{M}'$ if the following conditions hold:*

(i) For every $g \in I$, there is a $g' \in I'$ with $g \sim_s g'$;

Whenever $g \sim_s g'$, then

(ii) $V(g) = V'(g')$;

(iii) If $\mathcal{K}_i(g, g^1)$ for some $g^1 \in G$ and some $i \in \mathcal{A}$, then there is a $g'^1 \in G'$ such that $\mathcal{K}'_i(g', g'^1)$ and $g^1 \sim_s g'^1$.

(iv) If $(g, g^1) \in R$ for some $g^1 \in G$, then there is a $g'^1 \in G'$ such that $(g', g'^1) \in R'$ and $g^1 \sim_s g'^1$;

We say that a model $\mathcal{M}'$ simulates a model $\mathcal{M}$, denoted $\mathcal{M} \leq_s \mathcal{M}'$, if there is a simulation relation between $\mathcal{M}$ and $\mathcal{M}'$. ACTL*K formulae are preserved under simulation.

Theorem 2.1 ([CDLR09])**.** *Let $\mathcal{M}$ and $\mathcal{M}'$ be two models with $\mathcal{M} \leq_s \mathcal{M}'$. Suppose that $\mathcal{M}' \models \phi$ for some ACTL*K formula ϕ. Then, $\mathcal{M} \models \phi$.*

We call two models $\mathcal{M}$ and $\mathcal{M}'$ *simulation equivalent* if $\mathcal{M} \leq_s \mathcal{M}'$ and $\mathcal{M}' \leq_s \mathcal{M}$. The following theorem relates ACTL*K with simulation equivalence.

Theorem 2.2 ([CDLR09])**.** *Let $\mathcal{M}$ and $\mathcal{M}'$ be two simulation equivalent models. Then, $\mathcal{M} \models \phi$ iff $\mathcal{M}' \models \phi$, for any ACTL*K formula ϕ.*

A relation $\sim_s$ is a *bisimulation* if both $\sim_s$ and $\sim_s^{-1} = \{(g', g) \in G' \times G \colon g \sim_s g'\}$ are simulations. We define two models $\mathcal{M}$ and $\mathcal{M}'$ to be *bisimulation equivalent* if there is a bisimulation between $\mathcal{M}$ and $\mathcal{M}'$. The following theorem relates CTL*K with bisimulation equivalence.

Theorem 2.3 ([CDLR09])**.** *Let $\mathcal{M}$ and $\mathcal{M}'$ be two bisimulation equivalent models. Then, $\mathcal{M} \models \phi$ iff $\mathcal{M}' \models \phi$, for any CTL*K formula ϕ.*

2.3.2 Equivalences preserving ACTL*K$\backslash X$ and CTL*K$\backslash X$

Stuttering simulations are relations between models that preserve the satisfaction of stuttering-insensitive modal formulae. A model $\mathcal{M}' = (G', I', R', (\mathcal{K}'_i)_{i \in \mathcal{A}}, V')$ stuttering simulates a model $\mathcal{M} = (G, I, R, (\mathcal{K}_i)_{i \in \mathcal{A}}, V)$ if for every behaviour of the latter, there is stuttering equivalent behaviour of the former. Informally, two behaviours are stuttering equivalent if the behaviours coincide when each sequence of stutter steps (i.e., steps that do not affect the labelling of states) are collapsed onto a single step. Since an ACTL*K$\backslash X$ formula may only quantify over all paths, every ACTL*K formula satisfied by $\mathcal{M}$ is also satisfied by $\mathcal{M}'$.

Definition 2.10 (Stuttering simulation [LPQ10]). *A relation* $\sim_{ss} \subseteq G \times G'$ *is a* stuttering simulation *between* $\mathcal{M}$ *and* $\mathcal{M}'$ *if the following conditions hold:*

(i) *For every* $g \in I$*, there is a* $g' \in I'$ *with* $g \sim_{ss} g'$*;*

Whenever $g \sim_{ss} g'$*, then*

(ii) $V(g) = V'(g')$*;*

(iii) *If* $\mathcal{K}_i(g, g^1)$ *for some* $g^1 \in G$ *and some* $i \in \mathcal{A}$*, then there is a* $g'^1 \in G'$ *such that* $\mathcal{K}'_i(g', g'^1)$ *and* $g^1 \sim_{ss} g'^1$*.*

(iv) *For every path* $\pi \in \Pi(g)$*, there is a path* $\pi' \in \Pi(g')$*, a partition* $B_1, B_2, \ldots$ *of the states in* π*, and a partition* $B'_1, B'_2, \ldots$ *of the states in* π' *such that for each* $j \geq 1$*,* B_j *and* B'_j *are nonempty and finite, and every state in* B_j *is related by* $\sim_{ss}$ *to every state in* B'_j*.*

We say that a model $\mathcal{M}'$ stuttering simulates a model $\mathcal{M}$, denoted $\mathcal{M} \leq_{ss} \mathcal{M}'$, if there is a stuttering simulation relation between $\mathcal{M}$ and $\mathcal{M}'$. ACTL*K$\backslash X$ formulae are preserved under stuttering simulation.

Theorem 2.4 ([LPQ10]). *Let* $\mathcal{M}$ *and* $\mathcal{M}'$ *be two models with* $\mathcal{M} \leq_{ss} \mathcal{M}'$*. Suppose that* $\mathcal{M}' \models \phi$ *for some ACTL*K*$\backslash X$ *formula* ϕ*. Then,* $\mathcal{M} \models \phi$*.*

We call two models $\mathcal{M}$ and $\mathcal{M}'$ *stuttering simulation equivalent* if $\mathcal{M} \leq_{ss} \mathcal{M}'$ and $\mathcal{M}' \leq_{ss} \mathcal{M}$. The following theorem relates ACTL*K$\backslash X$ with stuttering simulation equivalence.

Theorem 2.5 ([LPQ10]). *Let* $\mathcal{M}$ *and* $\mathcal{M}'$ *be two stuttering simulation equivalent models. Then,* $\mathcal{M} \models \phi$ *iff* $\mathcal{M}' \models \phi$*, for any ACTL*K*$\backslash X$ *formula* ϕ*.*

A relation $\sim_{ss}$ is a *stuttering bisimulation* if both $\sim_{ss}$ and $\sim_{ss}^{-1} = \{(g', g) \in G' \times G \colon g \sim_s g'\}$ are stuttering simulations. We define two models $\mathcal{M}$ and $\mathcal{M}'$ to be *stuttering bisimulation equivalent* if there is a stuttering bisimulation between $\mathcal{M}$ and $\mathcal{M}'$. The following theorem relates CTL*K$\backslash X$ with stuttering bisimulation equivalence.

Theorem 2.6 ([LPQ10]). *Let $\mathcal{M}$ and $\mathcal{M}'$ be two stuttering bisimulation equivalent models. Then, $\mathcal{M} \models \phi$ iff $\mathcal{M}' \models \phi$, for any CTL*$K\backslash X$ formula ϕ.*

2.4 Model checking multiagent systems

Given a model M_S representing a system S and a formula ϕ_P encoding a specification P, *model checking* is the problem of establishing whether or not $M_S \models \phi_P$. Though M_S can be a model for the any logic, for the purpose of this thesis, M_S denotes either an interpreted system or an interleaved interpreted system, and ϕ_P denotes a temporal-epistemic formula.

This section briefly summarises some of the existing approaches in model checking multiagent systems. The techniques reviewed below extend techniques originally put forward for model checking temporal models by taking into account epistemic specifications. They are here categorised into four groups: (i) symbolic model checking; (ii) SAT-based translations; (iii) state space reductions; and (iv) automata-based techniques.

2.4.1 Symbolic model checking

The labelling algorithm

Given a model $\mathcal{M} = (G, I, R, (\mathcal{K}_i)_{i \in \mathcal{A}}, V)$ and a CTLK formula ϕ, the labelling algorithm calculates the set $[\![\phi]\!] = \{g \in G \colon g \models \phi\}$ of states satisfying ϕ. To do this, the algorithm works inductively on the structure of ϕ and labels the states in G with the sub-formulas of ϕ that are satisfied in those states. The computation is initiated on the smallest sub-formulas of ϕ and worked outwards toward ϕ in the following way:

- If ϕ is an atomic formula p, then label with p any state g such that $p \in V(g)$;
- If ϕ is of the form $\phi_1 \vee \phi_2$, then label with $\phi_1 \vee \phi_2$ any state g that is already labelled with either ϕ_1 or ϕ_2;
- If ϕ is of the form $\neg\phi_1$, then label with $\neg\phi_1$ any state g that is not already labelled with ϕ_1;

- If ϕ is of the form $E(\phi_1 U \phi_2)$, then:
  - label with $E(\phi_1 U \phi_2)$ any state g that is already labelled with ϕ_2;
  - repeat the following until no more states can be labelled: label with $E(\phi_1 U \phi_2)$ any state g that is already labelled with ϕ_1 and has at least one its R-related states labelled with $E(\phi_1 U \phi_2)$.
- If ϕ is of the form $EX\phi_1$, then label with $EX\phi_1$ any state g that has at least one of its R-related states labelled with ϕ_1.
- If ϕ is of the form $K_i\phi_1$, then label with $K_i\phi_1$ any state g that has all of its $\mathcal{K}_i$-related states labelled with ϕ_1.

Having calculated $[\![\phi]\!]$, the model checking problem can be solved by deciding whether or not $I \subseteq [\![\phi]\!]$. The algorithm can be analysed by means of the fixpoint characterisation of CTLK to show both termination and soundness [HR00, CGP99, RL05]. Its complexity is $|\phi| \, . \, (|G| + |R|)$, where $|\phi|$ is the number of connectives in ϕ, $|G|$ is the number of states, and $|R|$ is the number of transitions.

Symbolic model checking

Note that the complexity of the labelling algorithm is linear in the size of the model to check. Still, the size of the model is exponential in the number of variables encoding the system. This gives rise to the *state explosion problem*. To alleviate this problem, *symbolic model checking* techniques encode the model *symbolically* by means of compact data structures. This section discusses symbolic model checking using *Ordered Binary Decision Diagrams* (OBDDs).

OBDDs are a compact encoding for the manipulation of boolean functions. For example, assume the boolean function $f(a, b, c) \triangleq a \wedge (b \vee c)$. Its binary decision tree is depicted in Figure 2.2a. An algorithmic simplification of the binary decision tree results in the decision diagram shown in Figure 2.2b. This “reduced” structure is called the OBDD of f. OBDDs are canonical (i.e., each boolean function is represented by a unique OBDD) and can implement

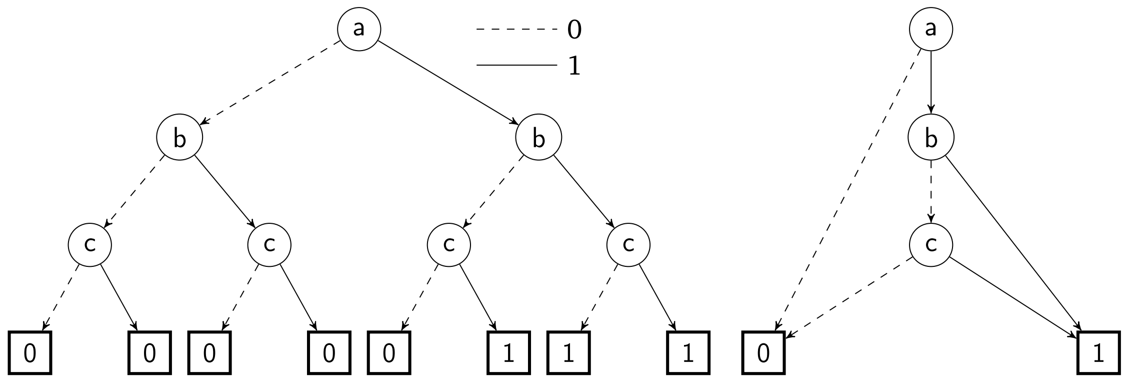


Figure 2.2: The OBDD representation of the boolean function $a \wedge (b \vee c)$ (from [Rai06]).

any operation on boolean functions in polynomial time. This makes them a valuable tool in digital system design, verification, and testing [Bry92].

In symbolic model checking, the states and relations of the model are encoded by means of boolean formulae. In this setting, the formula ϕ to check corresponds to the boolean formula representing the set of states satisfying ϕ. To compute it, the operations on sets used by the labelling algorithm are implemented in terms of operations on boolean functions. Finally, the latter are implemented in terms of operations on OBDDs. The interested reader is referred to [HR00, CGP99] for more details.

2.4.2 SAT-based translations

SAT-based translation techniques reduce the model checking problem to the boolean satisfiability problem (SAT). The exploitation of SAT solvers has advantages over symbolic model checking since SAT solvers do not usually require exponential space. The SAT-based translation techniques discussed below are: (i) *bounded model checking*; and (ii) *unbounded model checking*.

Bounded model checking

A logic $\mathcal{L}$ is said to have *bounded semantics* if the following holds: whenever a formula $\phi \in \mathcal{L}$ is satisfiable on a model $\mathcal{M}$, ϕ is satisfiable on $\mathcal{M}$ along a path of length k, for some integer

k. Instead of considering the entirety of the state space, bounded model checking techniques check the negation of ϕ on a fragment $\mathcal{M}^k$ of $\mathcal{M}$. Roughly speaking, $\mathcal{M}^k$ is obtained by taking all possible paths of length k in $\mathcal{M}$. Assuming the logic under consideration has bounded semantics, $\mathcal{M} \models \neg\phi$ iff there is an integer k such that $\mathcal{M}^k \models \neg\phi$. The latter decision problem is reduced to the satisfiability problem of the boolean formula $[\mathcal{M}]_k \wedge [\neg\phi]_k$, where $[\mathcal{M}]_k$ is the boolean encoding of $\mathcal{M}^k$ and $[\neg\phi]_k$ is the boolean encoding of $\neg\phi$. In other words, $\mathcal{M}^k \models \neg\phi$ iff $[\mathcal{M}]_k \wedge [\neg\phi]_k$ is satisfiable.

Bounded model checking was introduced in [BCCZ99] where the approach was formulated on LTL models. Bounded model checking for the universal fragment of CTL models was introduced in [PWZ02]. The ideas developed in the latter work were extended to the universal fragment of CTLK [ŁŁP03]. This line of work was followed by bounded model checking techniques for correct behaviour [WLP05], and bounded model checking techniques for knowledge in real time [LPW07].

Bounded model checking is remarkably more efficient than symbolic model checking in identifying counterexamples [3]. However, bounded model checking is an incomplete approach to model checking; i.e., it is not guaranteed to establish whether or not $\mathcal{M} \models \phi$. As a result, bounded model checking techniques are commonly used to find errors in the system under consideration, whereas symbolic model checking techniques are commonly used to show correctness. Therefore, each approach complements the other.

Bounded model checking techniques for multiagent systems are implemented in the tool `VerICS`. For a more detailed comparison of bounded and symbolic model checking knowledge, we refer to [KLN+06].

Unbounded model checking

Unbounded model checking is a symbolic model checking approach introduced in [McM02]. In this approach, the set of states and the transition relations are encoded in conjunctive normal form instead of OBDDs. This representation makes it possible to reduce the operations defined for symbolic model checking to boolean satisfiability problems. Unbounded model checking

[3]A counterexample is a path on which the negation of the formula to check is satisfied.

was adapted to knowledge in [KLP04] and [KP04a]. The former work deals with combinations of temporal and epistemic logic, whereas the latter work deals with combinations of alternating time logic and epistemic logic. Unbounded model checking techniques for multiagent systems are also implemented in the tool `VerICS`.

2.4.3 State space reductions

State space reduction techniques reduce the state space to check without altering the satisfaction status of a given formula. Typically, these techniques can be used in combination with either symbolic model checking or bounded model checking techniques. This section discusses *abstraction* and *partial order reduction*.

Abstraction

Abstraction techniques build an abstract, smaller model that is based on the original model. The abstract model is used instead of the original one to verify a given formula. There are different ways to abstract the model. Below, *existential abstraction* and *symmetry reduction* are discussed.

Existential abstraction. An existential abstraction is a surjective function $h : G \to \hat{G}$ that maps the set G of concrete states to the set $\hat{G}$ of abstract states. By means of h, the concrete state space is partitioned into equivalence classes making up the abstract state. In other words, the abstract space is equal to $\hat{G} = \{[g] \colon g \in G\}$, where $[g]$ denotes the equivalence class containing the concrete state g. The equivalence classes are induced by the equivalence relation $\{(g, g') \colon h(g) = h(g')\}$. In existential abstraction, an abstract model is built that simulates the concrete model, thus preserving the satisfaction of universal formulae from the abstract model to the concrete one.

Existential abstraction is applied to interpreted systems in [CDLR09]. Given an interpreted system $IS = \big((L_i, Act_i, P_i, t_i)_{i \in \mathcal{A}}, I, V\big)$, the local states and actions of each agent are collapsed into equivalence classes. The equivalence relations are chosen by the user. It is assumed that the atomic propositions do not distinguish between equivalent local states, i.e., for every pair

(g, g') of concrete global states, if $p \in V(g)$ and $ls_i(g) \equiv ls_i(g')$ for all $i \in \mathcal{A}$, then $p \in V(g')$. The abstract interpreted system $\hat{IS} = \left(\left(\hat{L}_i, \hat{Act}_i, \hat{P}_i, \hat{t}_i\right)_{i \in \mathcal{A}}, \hat{I}, \hat{V}\right)$ is built as follows.

- $\hat{L}_i = \{[l] \colon l \in L_i\}$;
- $\hat{Act}_i = \{[a] \colon a \in Act_i\}$;
- $\hat{P}_i([l]) = \{[a] \colon a \in P_i(l)\}$;
- $\hat{t}_i([l], [\overline{a}]) = [l']$ iff $t_i(l, \overline{a}) = l'$;
- $\hat{V}([g]) = V(g)$.

It is shown that $\hat{I}$ simulates I. $\hat{I}$ can therefore be used instead of I for the verification of an ACTLK formula.

Symmetry reduction. Intuitively, *symmetry* is an interchange of certain objects, e.g., local states, in the system that preserves the behaviour of the system. Formally, a symmetry of a set $X \subseteq G$ of global states is a bijection $\zeta : G \to G$ that does not alter X, i.e, $\zeta(X) = \{\zeta(g) \colon g \in X\} = X$. Similarly, a symmetry of a relation $X \subseteq G \times G$ between global states is a bijection $\zeta : G \to G$ that does not alter X, i.e., $\zeta(X) = \{(\zeta(g), \zeta(g')) \colon (g, g') \in X\}$.

Symmetries in multiagent systems have been exploited by means of *data symmetries* [CDLQ09a]. Data symmetries are domain permutations for the variables encoding the system: $\zeta_v : D_v \to D_v$, where D_v is the domain of variable v. Bijections can be defined on local and global states by point-wise application of ζ on the data elements occurring in the states. Two global states g, g' are said to be *data symmetric*, denoted $g \equiv g'$, iff $\zeta(g) = g'$ for some domain permutation ζ. Given an interpreted system $IS = \left((L_i, Act_i, P_i, t_i)_{i \in \mathcal{A}}, I, V\right)$, it is shown that a CTLK formula can be equivalently evaluated on the system $\hat{IS} = \left((L_i, Act_i, P_i, t_i)_{i \in \mathcal{A}}, \hat{I}, V\right)$, where $\hat{I}$ contains exactly one (representative) initial state for each equivalence class induced by the equivalence relation $\equiv$.

Additionally, symmetries in multiagent systems have been exploited by means of *agent symmetries* [CDLQ09b]. Agent symmetries are permutations on the agents encoding the system: $\zeta : \mathcal{A} \to \mathcal{A}$. Bijections on global states can defined by replacing the local state of each agent

i with the local state of the agent $\zeta(i)$. Two global states g, g' are said to be *agent symmetric*, denoted $g \equiv g'$, iff $\zeta(g) = g'$ for some agent permutation ζ. Given an interpreted system $IS = \left((L_i, Act_i, P_i, t_i)_{i \in \mathcal{A}}, I, V\right)$, it is shown that a CTLK formula can be equivalently evaluated on the system $\hat{IS} = \left((L_i, Act_i, P_i, t_i)_{i \in \mathcal{A}}, \hat{I}, V\right)$, where $\hat{I}$ contains exactly one (representative) initial state for each equivalence class induced by the equivalence relation $\equiv$.

Partial order reduction

Partial order reduction is a state space reduction technique for interleaved systems. Intuitively, an interleaved system allows for a sequence of actions to be performed in arbitrary orderings of the actions. Given a sequence of actions, this can be exploited to reduce the state space whenever the following two conditions are met: (i) the resulting global state is irrespective of the specific ordering of the actions in the sequence; (ii) the formula to check cannot distinguish between different orderings of the actions in the sequence. Assuming these conditions are met, partial order reduction techniques remove all orderings of a given path but one. This is typically accomplished by building the reduced model in a depth-first-search manner. The algorithm to do this selects only some of the possible paths to explore. Specifically, the algorithm eliminates paths for which a stuttering equivalent path already occurs in the reduced model.

The above ideas have been extended to interleaved interpreted systems in [LPQ10]. For a more detailed discussion on partial order reduction, the interested reader is referred to [Pel93].

2.4.4 Automata-based techniques

Automata-based techniques reduce the model checking problem to the nonemptiness problem for Büchi automata. A Büchi automaton is a tuple $A = (Q, Q^0, \Sigma, \delta, F)$ where:

- Q is a finite set of states;
- $Q^0 \subseteq Q$ is a set of initial states;
- Σ is a finite alphabet;

- $\delta \subseteq Q \times \Sigma \times Q$ is a set of transitions;
- $F \subseteq Q$ is a set of accepting states.

Given an infinite word $\sigma = a_0, a_1, \ldots, \sigma \in \Sigma^*$, a *run* r of A on σ is a sequence $q_0, q_1, \ldots$, where $q_0 \in Q^0$ and $(q_i, a_i, q_{i+1}) \in \delta$, for all $i > 0$. Let $i(r)$ be the set of states that occur in r infinitely often. A Büchi automaton A is said to accept a run r if $i(r) \cap F \neq \emptyset$ (i.e., there is some accepting state in F that occurs infinitely often in r). A Büchi automaton A is said to accept an infinite word σ if there is an accepting run of A on σ. The language of A is the set of infinite words accepted by A; it is denoted by $\mathcal{L}(A)$. The nonemptiness problem for a Büchi automaton A is to decide whether the language of A is nonempty, i.e, $\mathcal{L}(A) \neq \emptyset$.

The model checking problem is reduced to the nonemptiness problem of Büchi automata in the following way. Let $\mathcal{M}$ be a temporal model and let ϕ be an LTL formula. $\mathcal{M}$ is translated into a Büchi automaton A_M such that $\mathcal{L}(A_M)$ corresponds to the set of paths in $\mathcal{M}$. Further, the formula $\neg\phi$ is translated into a Büchi automaton $A_{\neg\phi}$ such that $A_{\neg\phi}$ accepts precisely the paths satisfying $\neg\phi$. It is proven that $\mathcal{M} \models \phi$ iff $\mathcal{L}(A_m) \cap \mathcal{L}(A_{\neg\phi}) = \emptyset$.

The method was introduced in [VW86] for the logic LTL. It was extended to epistemic logic and interpreted systems with perfect recall [4] [MS99]. The model checker MCK [GvdM04] implements these theoretical results.

2.4.5 The model checker MCMAS

MCMAS is a model checker for the verification of multi-agent systems. It is implemented in C++ and released as open source. MCMAS supports symbolic model checking techniques for CTLK; OBDDs are manipulated using the CUDD library [Som05]. Additionally, the checker can handle specifications expressed in *alternating time logic* (ATL) [AHK02] and *deontic* logic [LS03]. MCMAS takes as input an ISPL file. An ISPL file is structured into four sections.

1. **Agents' declarations section.** The agents' declarations follow closely the modular framework of interpreted systems. In particular, an agent's declaration includes declarations

[4]Intuitively, in a system with perfect recall each agent knows its history of local states it has gone through [FHMV95].

of the agent's local states, its local actions, its protocol and its evolution function.

The set of local states is given in terms of a set of variables. Each variable can be of either one of three types: (i) integer; (ii) Boolean; (iii) enumeration.

The protocol is declared in terms of `condition:actions` pairs. The `condition` specifies the values of the variables enabling the `actions`.

The evolution function is described by means of `if-then` statements. The precondition is a boolean expression specifying the current values of the variables and the joint action, whereas the postcondition determines the variables' next values.

```
Agent <agent_id>
  <agent_body>
end Agent

agent_body::
  Vars:
    <var_ID> : <var_domain>;
    ...
  end Vars
  Actions = { <action_id> , <action_id> ... };
  Protocol:
    <protocol_condition> : { <action_id>, <action_id> ... };
    ...
  end Protocol
  Evolution:
    <assignment> if <precondition> ;
    ...
  end Evolution
```

2. **Evaluation section.** The valuation function is encoded as a list of `if-then` statements. The precondition is a boolean expression specifying the values of the variables for which the atomic proposition expressed by the postcondition holds.

```
Evaluation
  <proposition_id> if <boolean_expression>;
  ...
end Evaluation
```

3. **Initial states section.** The set of initial global states is represented by a boolean expression on the variables.

```
InitStates
  <boolean_expression>
end InitStates
```

4. **Formulae section.** The set of formulae to check are built as usual from the atomic propositions specified in the evaluation section.

```
Formulae
  <formulae_list>
end Formulae
```

Figure 2.3 shows the `ISPL` file encoding the Train-Gate-Controller.

2.5 Parameterised model checking

The parameterised model checking problem is the decision problem of establishing whether a certain specification holds on a system comprised of an arbitrary number of participants. The problem has been extensively studied in the context of reactive systems. In the general setting it was shown to be undecidable [AK86]. However, given the importance of the problem, a rich body of work has been mainly concerned with three ways to tackle its difficulty:

1. The identification of restricted classes of systems on which decidability results can be drawn. For instance, specific communication topologies were analysed, e.g., rings [EN95, EK04], trees [AHH13, AJMd02], cliques [DSZ11], linear topologies [AHH13, ADHR07], when analysing network protocols for an unbounded number of hosts.

```
Agent Environment
Vars:
   s: { g,r }
end Vars
Actions = { E_1,L_1,E_2,L_2 };
Protocol:
   s = g : { E_1,E_2 };
   s = r : { L_1,L_2 };
end Protocol
Evolution:
   s = g if s = r and ((Action = L_1) or (Action = L_2 and
                                           T_2.Action = L_2));
   s = r if s = g and ((Action = E_1) or (Action = L_2 and
                                           T_2.Action = E_2));
end Evolution
end Agent

Agent  T_1
Vars:
   s: { w,t,a }
end Vars
Actions = { E_1,L_1,B_1 };
Protocol:
   s = w : { E_1 };
   s = t : { L_1 };
   s = a : { B_1 };
end Protocol
Evolution:
   s = w if s = a and Action = B_1 ;
   s = t if s = w and Action = E_1 and Environment.Action = E_1;
   s = a if s = t and Action = L_1 and Environment.Action = L_1;
end Evolution
end Agent

Agent T_2
    ...
end Agent

Evaluation
    t_1 if T1.s = t;
    t_2 if T2.s = t;
end Evaluation

Formulae
     AG ( t_1 -> K(T_1,!t_2);
end Formulae
```

Figure 2.3: The ISPL description of the Train-Gate-Controller.

2. The development of sound but necessarily incomplete methodologies. These methodologies are not generally guaranteed to solve the parameterised model checking problem. Still, they have made it possible to formally verify a variety of algorithms including Szymanski's mutual exclusion algorithm [Szy88], Lamport's bakery algorithm [PXZ02, ADHR07], the dining philosophers protocol [HBR09], and several cache coherence protocols [HSBR10, ADHR07].

3. The development of partially automated methodologies [CTV06, CTV08, WL90, CGB89, CG87, KM89]. These techniques require manual guidance.

The majority of studies in parameterised model checking tackle safety properties [ACJT96, AHH13, Mai01, ADHR07, DT98]. Others target liveliness properties [EFM99, EN98, PXZ02, PS00, BJ$^+$00]. Some investigate LTL specifications [HBR09, EK00, HSBR10, GS92, EN96, EK03b], CTL specifications [CTTV04], and CTL* specifications [EN95, AJKR14, CGB89]. One issue in these works is to avoid formalisms that are able to represent the number of participants in the system. For example, consider a ring of processes that communicate by passing a token in a fixed direction, and assume that the proposition t_1 is true whenever process 1 has the token. Then, the formula

$$AG(t_1 \rightarrow XXXXXt_1)$$

is read "whenever process 1 has the token, it will again receive the token in exactly five steps". This is only true if the ring has exactly five processes [CGB89]. It is often the case that properties of this form lead the parameterised model checking problem to undecidability [EK03b]. The next operator X is therefore typically excluded from the specifications to avoid this.

The next two sections report on some of the ways the parameterised model checking problem is formalised and on the techniques employed to solve it.

2.5.1 Parameterised modelling languages

The parameterised model checking problem (PMCP) is typically formulated in a finitary, abstract way by giving a template for the participants in the system and the formula to be verified. By providing the parameter n representing number of actual participants in the system, the concrete system of n participants can be constructed upon which the standard model checking problem can be solved. In a widely studied definition [EK00, GS92, EN96, EN98, EK03b, EFM99, AKR+14], a template $\mathcal{T}$ is given by a 4-tuple $\mathcal{T} = (S, L, R, \iota)$, where

- S is a finite, nonempty set of states;
- L is a finite set of transition labels;
- ι is the initial state;
- $R \subseteq S \times L \times S$ is the transition relation.

The set L of transition labels is equal to the union of the sets Σ_{in} of internal transition labels, $\Sigma_b \times \{?\}$ of *input broadcast* labels, $\Sigma_b \times \{!\}$ of output *broadcast labels*, $\Sigma_{ar} \times \{?\}$ of *input asynchronous rendezvous* labels, $\Sigma_{ar} \times \{!\}$ of *output asynchronous rendezvous* labels, $\Sigma_{pr} \times \{?\}$ of *input pairwise rendezvous* labels, $\Sigma_{pr} \times \{!\}$ *output pairwise rendezvous* labels, and the sets of *conjunctive* and *disjunctive boolean guards*. These sets of labels correspond to different communication primitives studied in the literature:

- **Internal actions [GS92].** A participant performs an internal action with no communication taking place.

  Assume two global states g and g' of a given concrete system. A global transition from g to g' can happen by means of an internal label a if there is a participant i such that $(ls_i(g), a, ls_i(g')) \in R$, and for all participants $j \neq i$, $ls_j(g) = ls_j(g')$.

- **Broadcast [EN98].** A participant sends a message to all other participants. It is assumed that each participant is always ready to receive the message. The transition relation is therefore in compliance with the following condition: for every $a \in \Sigma_b$ and every state $s \in S$, there is a state $s' \in S$ such that $(s, a?, s') \in R$.

  Assume two global states g and g' of a given concrete system. A global transition from g to g' can happen by means of a broadcast label a if there is a participant i such that $(ls_i(g), a!, ls_i(g')) \in R$, and for all participants $j \neq i$, $(ls_j(g), a?, ls_j(g')) \in R$.

- **Asynchronous rendezvous [DRB02].** A participant sends a message irrespective of whether there is a participant ready to receive the message.

  Assume two global states g and g' of a given concrete system. A global transition from g to g' can happen by means of an asynchronous rendezvous label a if there is a participant i such that $(ls_i(g), a!, ls_i(g')) \in R$, and either: (i) for all participants $j \neq i$, $ls_j(g) = ls_j(g')$; i.e, there is no participant ready to receive the message; or (ii) there is a participant $j \neq i$ such that $(ls_j(g), a?, ls_j(g')) \in R$, and for all participants $q \neq j$, $q \neq i$, $ls_q(g) = ls_q(g')$; i.e., a participant is ready to receive the message.

- **Pairwise rendezvous [GS92].** A participant sends a message only if there is a participant ready to receive the message.

  Assume two global states g and g' of a given concrete system. A global transition from g to g' can happen by means of a pairwise rendezvous label a if the following hold: (i) there is a participant i such that $(ls_i(g), a!, ls_i(g')) \in R$; (ii) there is a participant $j \neq i$ such that $(ls_j(g), a?, ls_j(g')) \in R$; and (iii) for all participants $q \neq j$, $q \neq i$, $ls_q(g) = ls_q(g')$.

- **Conjunctive boolean guards [EK00].** A conjunctive boolean guard is constructed from the schema $\bigwedge(a \vee \ldots \vee b)$. A guard of this form expresses that all participants other than the one firing the transition are currently in one of the local states $a, \ldots, b$.

  Assume two global states g and g' of a given concrete system. A global transition from g to g' can happen by means of a conjunctive boolean guard $\bigwedge(a \vee \ldots \vee b)$ if there is a participant i such that $(ls_i(g), \bigwedge(a \vee \ldots \vee b), ls_i(g')) \in R$, and for all participants $j \neq i$, $ls_j(g) = ls_j(g') = a \vee \ldots \vee ls_j(g) = ls_j(g') = b$.

- **Disjunctive boolean guards [EK00].** A disjunctive boolean guard is constructed from the schema $\bigvee(a \vee \ldots \vee b)$. A guard of this form expresses that there is a participant other than the one firing the transition in one of the local states $a, \ldots, b$.

  Assume two global states g and g' of a given concrete system. A global transition from g to g' can happen by means of a disjunctive boolean guard $\bigvee(a \vee \ldots \vee b)$ if the following hold: (i) there is a participant i such that $(ls_i(g), \bigvee(a \vee \ldots \vee b), ls_i(g')) \in R$; (ii) there is a participant $j \neq i$ such that $ls_j(g) = a \vee \ldots \vee ls_j(g) = b$; and (iii) for all participants $q \neq i$, $ls_q(g) = ls_q(g')$.

Each of the above primitives has been separately studied, but always in combination with internal actions.

Broadcast protocols were introduced in [EN98]. The PMCP has been analysed in terms of safety, liveliness, and LTL properties. It was initially shown to be decidable for safety properties and undecidable for liveliness properties [EFM99]. It was then shown to be undecidable for LTL$\backslash X$ properties [EK03b]. Nevertheless, a decidability result was given for LTL properties under the restriction of *initialisable templates* [EK03b]; i.e, systems in which every state of the template has a transition by means of an internal label to the initial state.

Asynchronous rendezvous primitives were introduced in [DRB02] for the verification of multi-threaded Java programs. The PMCP was shown to be decidable for safety properties and undecidable for LTL$\backslash X$ properties [EK03b].

Pairwise rendezvous primitives were introduced in [GS92] where a parameterised model checking procedure for LTL properties was put forward.

Disjunctive and conjunctive guards first appeared in [EK00]. For both settings, the PMCP was shown to be decidable for LTL$\backslash X$ properties. It was later observed that the inclusion of the next operator leads to undecidability [EK03b].

Other parameterised modelling languages have been considered. In particular, there is an increasing interest in the verification of communication networks with an unbounded number of hosts. Typically, the communication topology of a network is represented by a graph. The graph may correspond to specific, predefined topologies [CTTV04, EN95, AJKR14], or to arbitrary topologies generated by context-free grammars [SG90, BCK08, SWJ08]. In a network grammar, an initial variable represents the network type, the terminals represent instances of process types, and the nonterminals represent subnetwork types [SG90]. The parameterised model checking problem in this context is the decision problem of determining whether a given specification holds irrespectively of the size of a generated topology.

$$\begin{tikzcd}
\text{Concrete space} & (\mathbf{n},\mathbf{n},\mathbf{l},\mathbf{o},\mathbf{l},\mathbf{n},\mathbf{n},\mathbf{n},\mathbf{l},\mathbf{l}) \arrow[r] \arrow[d, dotted] & (\mathbf{l},\mathbf{o},\mathbf{l},\mathbf{n},\mathbf{l},\mathbf{n},\mathbf{l},\mathbf{n},\mathbf{l},\mathbf{l}) \arrow[d, dotted] \\
\text{Abstract space} & ((\mathbf{l},\mathbf{4}),(\mathbf{m},\mathbf{0}),(\mathbf{n},\mathbf{5}),(\mathbf{o},\mathbf{1})) \arrow[r] & ((\mathbf{l},\mathbf{6}),(\mathbf{m},\mathbf{0}),(\mathbf{n},\mathbf{3}),(\mathbf{o},\mathbf{1}))
\end{tikzcd}$$

Figure 2.4: Counter abstraction.

$$\begin{tikzcd}
\text{Concrete space} & (\mathbf{n},\mathbf{n},\mathbf{l},\mathbf{o},\mathbf{l},\mathbf{n},\mathbf{n},\mathbf{n},\mathbf{l},\mathbf{l}) \arrow[r] \arrow[d, dotted] & (\mathbf{l},\mathbf{o},\mathbf{l},\mathbf{n},\mathbf{l},\mathbf{n},\mathbf{l},\mathbf{n},\mathbf{l},\mathbf{l}) \arrow[dl, dotted] \\
\text{Abstract space} & ((\mathbf{l},\mathbf{2}),(\mathbf{m},\mathbf{0}),(\mathbf{n},\mathbf{2}),(\mathbf{o},\mathbf{1})) \arrow[loop below] &
\end{tikzcd}$$

Figure 2.5: Counter abstraction with a threshold of 2.

2.5.2 Parameterised model checking techniques

This section briefly summarises some of the existing approaches in parameterised model checking. The techniques discussed below are categorised into three groups: (i) abstraction; (ii) cutoffs; and (iii) network invariants.

Abstraction

Abstraction techniques [PXZ02, CTV08, GS92, CTV06, EN96, EFM99, YL10, EN98] rely on the analysis of a single finite state *abstract system* encoding all possible concrete systems. Typically, these techniques require manual guidance for obtaining the abstract mapping. Further, they are often incomplete: if a certain specification is falsified on the abstract model, then it does not necessarily follow that there is a concrete system falsifying the specification. Below we discuss *counter abstraction* and *environment abstraction*.

Counter abstraction

Intuitively, in counter abstraction the identities of the participants are abstracted away. An abstract state is a tuple of counters, one per local state, such that it reflects the number of participants in each local state. Figure 2.4 exemplifies this idea on a global transition of a concrete system with ten participants. In the figure the concrete state $g = (n, n, l, o, l, n, n, n, l, l)$, representing that the first participant is in state n, the second is in state n, the third is in state l, and so forth, is abstracted to $\gamma = ((l, 4), (m, 0), (n, 5), (o, 1))$, representing that there are four participants in state l, zero participants in state m, and so forth. Similarly, the state $g' = (l, o, l, n, l, n, l, n, l, l)$ is abstracted to $\gamma' = ((l, 6), (m, 0), (n, 3), (o, 1))$. The concrete transition $g \to g'$ is abstracted to $\gamma \to \gamma'$.

Counter abstraction is explored in [GS92] where the abstract model is defined in terms of a Petri Net [KM69]. In this work a concrete system is built by templates with pairwise rendezvous labels. It is observed that since the number of participants is unbounded, the abstract space is unbounded. Nevertheless, it is shown that the PMCP can be solved by considering only the paths in the abstract model that are bounded in length. As each of these paths has a concrete representative path in a system of a bounded number of participants, an automata-theoretic procedure can be defined for the verification of LTL properties. The procedure runs in time doubly exponential in the size (the number of states and transitions) of the template from which a concrete system is generated.

In other approaches, the counters in the abstract model are abstracted by imposing a certain threshold on their values. Figure 2.5 exemplifies this on a global transition with ten participants and a threshold of 2. In the figure the concrete states $g = (n, n, l, o, l, n, n, n, l, l)$ and $g' = (l, o, l, n, l, n, l, n, l, l)$ are abstracted to $((l, 2), (m, 0), (n, 2), (o, 1))$, representing that there are at least two participants in state l, zero participants in state m, at least two participants in state n, and one participant in state o. Similarly, the state $g' = (l, o, l, n, l, n, l, n, l, l)$ is abstracted to $\gamma' = ((l, 6), (m, 0), (n, 3), (o, 1))$. The concrete transition $g \to g'$ is abstracted to $\gamma \to \gamma'$.

Although the threshold-based counter abstraction of a given system corresponds to a finite abstract model, it often does so by overapproximating concrete behaviours, thereby hindering the verification of a property. For example, as shown in Figure 2.5, there is a transition from the abstract state to itself, whereas the abstract state has no concrete representative with a transition to itself. A *spurious path* is said to be an abstract path that does not correspond to any concrete behaviour. These paths are identified via a procedure given in [EN96]. The procedure is given in the context of templates with conjunctive and disjunctive boolean guards, as presented in Section 2.5.1, but with synchronous composition of the participants. In the asynchronous case, the PMCP is shown undecidable for LTL formulae [EN96]. This was followed by an incomplete counter abstraction procedure for the verification of asynchronously communicating processes [PXZ02].

Environment abstraction

Environment abstraction combines counter abstraction with predicate abstraction. In environment abstraction, rather than keeping track of the number of participants in each local state, a record of the number of participants that satisfy certain predicates is kept. A predicate is a first order atomic formula over the variables encoding the template. The abstract transition relation in this setting is defined by a parametric extension of existential abstraction. In other words, there is an abstract transition from a state γ to a state γ' if there is a concrete system of $n \geq 1$ participants in which there is a concrete transition from a state g to a state g', where g, g' are represented by γ, γ', respectively. These ideas have been put forward in [CTV06, CTV08].

Cutoffs

Cutoff techniques identify an integer called *the cutoff*. A cutoff expresses the number of components that is sufficient to consider when evaluating a given specification. So, rather than checking every concrete system for every value of the parameter, the PMCP can be solved by checking all concrete systems up to the cutoff. The cutoff identification procedures may either be *dynamic* or *static*, as discussed below.

Dynamic cutoff identification procedures identify cutoffs on-the-fly during the verification procedure. At each step of the procedure, a concrete system of n participants is verified and it is in parallel determined whether or not n is a cutoff. If so, then the verification procedure terminates. Otherwise, the procedure proceeds to the next step where a concrete system of $n + 1$ participants is considered. Dynamic cutoffs were used for the analysis of boolean programs against safety properties [KKW10], linear and tree like topologies against safety properties [AHH13], and arbitrary topologies against LTL$\backslash X$ properties [HBR09, HSBR10].

Static cutoff identification procedures identify cutoffs before the actual verification commences. The cutoff procedure takes as input the participants' template and the formula to check, and returns a cutoff. Following the calculation of a cutoff, each concrete system up to the cutoff is built and verified. Differently from dynamic approaches, where the minimal cutoff is identified, static approaches do not necessarily identify a minimal cutoff. However, they can typically analyse richer specifications. Static cutoffs were used for the analysis of ring topologies against CTL$^* \backslash X$ properties [EN95], arbitrary topologies against CTL$\backslash X$ [CTTV04] and CTL$^* \backslash X$ properties [AJKR14], systems built from templates with conjunctive and disjunctive guards against LTL$\backslash X$ properties [EK00], and resource allocation networks against LTL$\backslash X$ properties [EK02].

Network invariants

Network invariant approaches assume the specification is represented by a finite state system. Satisfaction of the specification on a concrete system is established in terms of a preorder on systems. That is, a concrete system $\mathcal{T}(n)$ satisfies the specification ϕ iff $\mathcal{T}(n) \leq \phi$, for some preorder $\leq$. Therefore, the PMCP in this setting is to check whether or not $\mathcal{T}(n) \leq \phi$, for every $n \geq 1$. This is determined by identifying a *network invariant*. Informally, a network invariant I is a finite state system encoding any possible behaviour of any concrete system. Formally, I is a finite state system such that:

1. $\mathcal{T} \leq I$;
2. $\mathcal{T} \parallel I \leq I$, where $\parallel$ denotes parallel composition.

Note that if I is a network invariant, then $\underbrace{\mathcal{T} \parallel \mathcal{T} \parallel \cdots \parallel \mathcal{T}}_{n} \leq I$, i.e., $\mathcal{T}(n) \leq I$, for any $n \geq 1$. The PMCP is thus reduced to the problem of determining whether or not $I \leq \phi$.

The method of network invariants was introduced in [WL90]. It is shown that invariants do not generally exist. Investigations on network invariants were followed in [AJ99], where sufficient criteria for their existence were given. Heuristics for the automatic generation of invariants were later introduced in [APR$^+$01, ZP04].

Chapter 3

Formalisms for unbounded multiagent systems

This chapter proposes two formalisms for reasoning about unbounded multiagent systems. Section 3.1 introduces a semantics for synchronous UMAS that extends interpreted systems. Section 3.2 presents a semantics for asynchronous UMAS that is based on interleaved interpreted systems. Section 3.3 introduces a specification language that extends temporal-epistemic logic, and that can be used to express properties of agents in a UMAS. Section 3.4 follows with the definition of certain notions of simulations between concrete systems, and an analysis on the preservation of logical satisfaction between similar systems.

3.1 Parameterised interpreted systems

This section introduces *parameterised interpreted systems* (PIS) [KL15b]. PIS extend interpreted systems to reason about temporal-epistemic properties in a UMAS setting. A PIS models an unbounded collection of agents where each agent adheres to a different *role*. Each role is associated with a generic *agent template* which specifies the behaviour of each agent of said role. The description of a PIS consists of the descriptions of a finite number of agent templates and the description of the *environment template*. The environment template expresses the behaviour of the environment under any given number of agents. A parameter for a PIS is a tuple of natural numbers, one for each role, that determines the actual number of agents in the system. Given a parameter $(n_1, \ldots, n_k)$ for the system, the concrete interpreted system corresponding to the composition of n_i agents, for each role i, can be constructed.

Assume a set $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ of agent templates and an environment template $\mathcal{E}$. Each template has a similar description to a standard agent in interpreted systems. However, its transition function does not depend on the joint action performed in the system, but it depends: (i) on the action performed by a concrete agent following this template; on the action of the environment; and (iii) on the projection of the joint action for all other concrete agents into a set. Thus, the identities of the agents are abstracted away in a joint action, thereby reflecting the unbounded nature of the system. In other words, a concrete agent can observe the actions performed in the system at a given time but it cannot observe which agent performed which action.

Definition 3.1 (Agent template). *An* agent template $\mathcal{T}_i$ *is a tuple* $\mathcal{T}_i = (L_i, Act_i, P_i, t_i)$*, where*

- L_i *is a set of template states;*
- Act_i *is a set of template actions;*
- $P_i : L_i \to \mathcal{P}(Act_i)$ *is the template protocol;*
- $t_i : L_i \times Act_i \times \mathcal{P}(\bigcup_{\mathcal{T}_i} Act_i) \times Act_{\mathcal{E}} \to L_i$ *is the template transition function.*

The environment template $\mathcal{E}$ is defined along the same lines as an agent template. Its transition function depends: (i) on the action performed by the environment; and (ii) on the projection of the joint action for all the concrete agents into a set.

Definition 3.2 (Environment template). *An* environment template $\mathcal{E}$ *is defined as a tuple* $\mathcal{E} = (L_{\mathcal{E}}, Act_{\mathcal{E}}, P_{\mathcal{E}}, t_{\mathcal{E}})$*, where*

- $L_{\mathcal{E}}$ *is a set of template states;*
- $Act_{\mathcal{E}}$ *is a set of template actions;*
- $P_{\mathcal{E}} : L_{\mathcal{E}} \to \mathcal{P}(Act_{\mathcal{E}})$ *is the template protocol;*

- $t_{\mathcal{E}} : L_{\mathcal{E}} \times Act_{\mathcal{E}} \times \mathcal{P}(\bigcup_{\mathcal{T}_i} Act_i) \to L_{\mathcal{E}}$ *is the template transition function.*

A PIS consists of a finite collection of agent templates and a template environment.

Definition 3.3 (Parameterised interpreted system). *A* Parameterised Interpreted System *is a tuple $PIS = (\mathcal{T}, \mathcal{E}, \mathcal{I}, \mathcal{V})$, where*

- $\mathcal{T} = \{(L_1, Act_1, P_1, t_1), \ldots, (L_{|\mathcal{T}|}, Act_{|\mathcal{T}|} P_{|\mathcal{T}|}, t_{|\mathcal{T}|})\}$ *is a nonempty and finite set of agent templates;*
- $\mathcal{E} = (L_{\mathcal{E}}, Act_{\mathcal{E}}, P_{\mathcal{E}}, t_{\mathcal{E}})$ *is an environment template;*
- $\mathcal{I} = I_1 \times \ldots \times I_{|\mathcal{T}|} \times I_{\mathcal{E}}$ *is the set of initial template states, where $I_i \subseteq L_i$ is the set of initial template states for agent template i.*
- $\mathcal{V} = \{V_i : L_i \to \mathcal{P}(AP_i) \colon i \in \mathcal{T}\}$ *is a set of valuation functions, one for each agent template. Each V_i is defined on a set of atomic propositions AP_i. It is assumed that $AP_1, \ldots, AP_{|\mathcal{T}|}$ are pairwise disjoint sets.*

Let $PIS = (\mathcal{T}, \mathcal{E}, \mathcal{I}, \mathcal{V})$ be a parameterised interpreted system of k roles. Let $\overline{n} = (n_1, \ldots, n_k) \in \mathbb{N}^k$ be a concrete value of the system's parameter. We write $\overline{n}.i$ for the natural number associated with template $\mathcal{T}_i$ in $\overline{n}$. For a k-tuple of natural numbers $\overline{x}$, we write $\overline{x} \leq \overline{n}$ to mean that $\overline{x}.1 \leq \overline{n}.1, \ldots \overline{x}.k \leq \overline{n}.k$. We now describe the $\overline{n}$'st concrete instantiation of a PIS. The concrete system $PIS(\overline{n})$ is an interpreted system that results from the composition of $\overline{n}.i$ instantiations $\{(i,1), \ldots, (i, \overline{n}.i)\}$ of each agent template $\mathcal{T}_i$, and an instantiation of the environment template. Given a $\overline{x} \leq \overline{n}$, we write $\mathcal{A}(\overline{x})$ for the set $\mathcal{A}(\overline{x}) = \{(i,j) \colon 1 \leq i \leq k, 1 \leq j \leq \overline{x}.i\}$ of concrete agents. The concrete agents are instantiated by taking indexed copies of their corresponding agent templates.

Definition 3.4 (Concrete agent). *Given a PIS $PIS = (\mathcal{T}, \mathcal{E}, \mathcal{I}, \mathcal{V})$ and a value $\overline{n} \in \mathbb{N}^k$ for its parameter, the concrete agent $(i,j) = \left(L_i^j, Act_i^j, P_i^j, t_i^j\right)$ is defined as follows:*

- $L_i^j = L_i \times \{j\}$*;*
- $Act_i^j = Act_i$*;*

- $P_i^j : L_i^j \to \mathcal{P}(Act_i^j)$ *is defined for each concrete local state* (l, j) *by* $P_i^j((l, j)) = P_i(l)$;
- $t_i^j : L_i^j \times ACT \to L_i^j$ *is such that*

$$t_i^j\left((l,j),\overline{a}\right) = (l',j) \text{ iff } t_i\left(l, la_i^j(\overline{a}), \{la_r^s(\overline{a})\colon (r,s) \in \mathcal{A}(\overline{n})\}, la_E(\overline{a})\right) = l',$$

where $la_i^j(\overline{a})$ *denotes the local action of agent* (i, j) *in* $\overline{a}$.

So, the local states of a concrete agent are made of the template local states indexed by the name of the agent in question. The agent inherits from its template the actions, the protocol and the transition function. The concrete environment is similarly obtained.

Definition 3.5 (Concrete environment). *Given a PIS* $PIS = (\mathcal{T}, \mathcal{E}, \mathcal{I}, \mathcal{V})$ *and a value* $\overline{n} \in \mathbb{N}^k$ *for its parameter, the concrete environment* $E = (L_E, Act_E, P_E, t_E)$ *is defined as follows.*

- $L_E = L_{\mathcal{E}}$;
- $Act_E = Act_{\mathcal{E}}$;
- $P_E : L_E \to \mathcal{P}(Act_E)$ *is defined for each concrete local state* l *by* $P_E(l) = P_{\mathcal{E}}(l)$;
- $t_E : L_E \times ACT \to L_E$ *is such that*

$$t_E\left(l, \overline{a}\right) = l' \text{ iff } t_{\mathcal{E}}\left(l, la_E(\overline{a}), \{la_r^s(\overline{a})\colon (r,s) \in \mathcal{A}(\overline{n})\}\right) = l',$$

where $la_r^s(\overline{a})$ *denotes the local action of agent* (r, s) *in* $\overline{a}$.

Finally, a PIS's instantiation, and the concrete semantics we consider for synchronous UMAS, is the interpreted system composed of the concrete agents and the concrete environment.

Definition 3.6 (Concrete system). *Given a PIS* $PIS = (\mathcal{T}, \mathcal{E}, \mathcal{I}, \mathcal{V})$ *and a value* $\overline{n} \in \mathbb{N}^k$ *for its parameter, the* concrete system $PIS(\overline{n})$ *is the interpreted system*

$$PIS(\overline{n}) = \left((L_i^j, Act_i^j, P_i^j, t_i^j)_{(i,j)\mathcal{A}(\overline{n})}, (L_E, Act_E, P_E, t_E), I, V\right),$$

where:

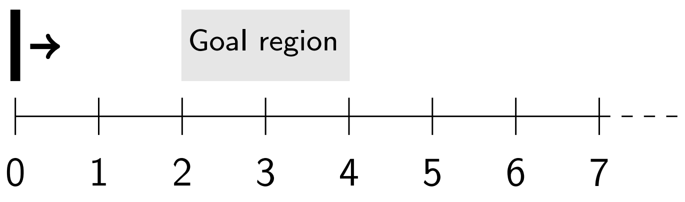


Figure 3.1: The autonomous robot scenario.

- $I = \{(l_1^1, \ldots, l_k^{\overline{n}.k}, l_E): l_1^1 \in I_1 \times \{1\}, \ldots, l_k^{\overline{n}.k} \in I_k \times \{\overline{n}.k\}, l_E \in I_\mathcal{E}\}$ *is the set of initial concrete states;*
- $V : S \to \mathcal{P}(AP)$ *is the concrete valuation function defined on the set* $S = L_1^1 \times \ldots \times L_k^{\overline{n}.k} \times L_E$ *of possible global states, and on the set* $AP = (AP_1 \times \{1, \ldots, \overline{n}.1\}) \cup \ldots \cup (AP_k \times \{1, \ldots, \overline{n}.k\})$ *of atomic propositions as follows:*

  for $p \in AP_i$ *and* $1 \leq j \leq \overline{n}.i, (p, j) \in V(g)$ *iff* $p \in V_i(l)$,

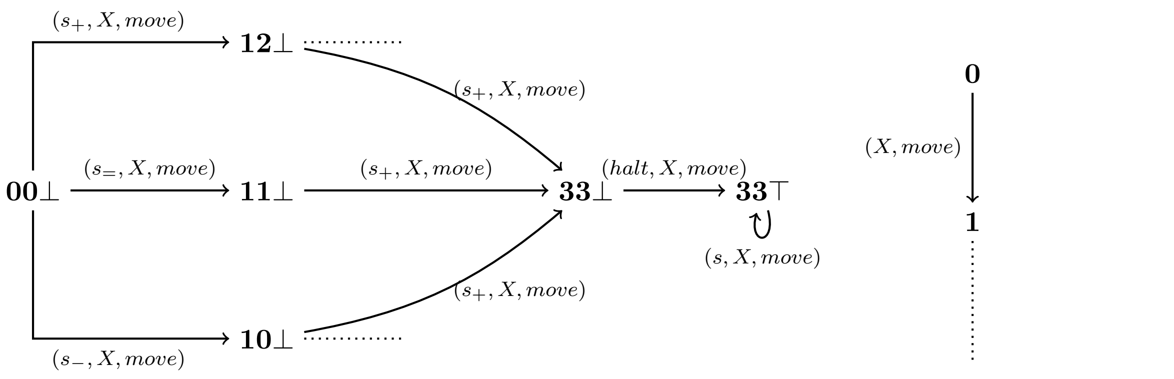
  where l *is the template local state of agent* (i, j) *in* g.

Hence a PIS $\mathcal{S}$ is a concise description of an unbounded collection $\{\mathcal{S}(\overline{n}): \overline{n} \in \mathbb{N}^k\}$ of (concrete) interpreted systems. For each concrete system $PIS(\overline{n})$, we can associate a temporal-epistemic model $\mathcal{S}_{PIS(\overline{n})} = \left(G(\overline{n}), I(\overline{n}), R(\overline{n}), (\mathcal{K}_i^j)_{(i,j) \in \mathcal{A}(\overline{n})}, V(\overline{n})\right)$ as described in section 2.1.1. When $PIS(\overline{n})$ is clear from the context we simply write $\mathcal{S}(\overline{n})$ for $\mathcal{S}_{PIS(\overline{n})}$. Given a global state g, we write $ls_i^j(g)$ for the local state of agent (i, j) in g. The template local state of agent (i, j) in g is denoted by $tls_i^j(g)$.

3.1.1 Autonomous robot

This section encodes a parameterised variant of the autonomous robot scenario in the parameterised interpreted systems language.

The autonomous robot scenario was introduced in [FHMV95]. It includes an autonomous robot running along an endless straight track (see Figure 3.1). The position of the robot is given in terms of locations numbered as $0, 1, 2, \ldots$. The robot is initially at position 0. It can only move forward along the track and its movement is controlled by the environment. The



(a) Template robot 1. (b) Environment template.

Figure 3.2: The parameterised interpreted system of the autonomous robots.

only action the robot can perform is to halt. If the robot halts, then the environment can no longer move the robot. Otherwise, the environment moves the robot one position forward at each time step.

A sensor is attached to the robot that measures its position. The sensor is faulty in the sense that a sensor reading at position q can be any of the values in $\{q-1, q, q+1\}$. The goal of the robot is to halt in the goal region $\{2, 3, 4\}$. A solution to the autonomous robot problem in the single robot case is for the robot to do nothing while the value of its sensor is less than 3, and to halt once the value of its sensor is greater than or equal to 3 [FHMV95]. We show that this solution applies to the arbitrary case with an unbounded number of robots.

We consider a generalisation of the above description in which an arbitrary number of robots run synchronously along the track. We encode this scenario as a PIS composed of an agent template representing the robots and an environment template modelling the environment. Since the templates can only be finite structures, the modelling here proposed assumes a finite track.

The template for the robots and the template for the environment are given by Figure 3.2. In the figure, an edge from node l to node l' that is labelled with (a, A, a_E) represents the template transition $t(l, a, A, a_E) = l'$. A template state (p, s, h) of a robot represents its position p, its sensor reading s, and whether it has halted ($h = \top$) or not ($h = \bot$). A template state p of the environment represents the position of the robots. The system has a unique initial template state $(00\bot, 0)$. The actions $s_=, s_+, s_-$ determine the status of the sensor reading at each time step. The status of a reading can either be: (i) correct whenever $s_=$ is performed; (ii) wrong and one position higher whenever s_+ is performed; or (iii) wrong and one position lower whenever s_- is performed. The symbol $X \subseteq \{s_+, s_-, s_+, s, halt\}$ denotes an arbitrary subset of the set of actions admitted by the template robot.

So, at each time step, the environment performs the action $move$ and moves to the next position in the track irrespective of agents' actions performed at the time step. Each agent non-deterministically performs one of the actions $s_=, s_+, s_-$ at each time step. Upon performing one of these actions, an agent moves to a state encoding the next position, say q, in the track, and the updated value of its sensor which could either be $q - 1$, q, or $q + 1$, depending on which action the agent performed; this transition is irrespective of the other agents' actions. Whenever the value of the sensor of an agent is greater than or equal to 3, the agent can only perform the $halt$ action; then, the agent updates the halting status in its state and it forever remains in the resulting local state.

Figure 3.3 depicts a fragment of the concrete system with two robots. Each global state is a 3-tuple representing, from left to right, the local state of the first robot, the local state of the second robot, and the local state of the environment. An edge from a global state g to a global state g' that is labelled with (a, a', a'') represents that robot 1 performs the action a in g, robot 2 performs the action a' in g, the environment performs the action a'' in g, and the resulting global state is g'.

3.2 Parameterised interleaved interpreted systems

This section introduces *parameterised interleaved interpreted systems* (PIIS) [KL13a, KL13b, KL15a, KL16]. PIIS extend interleaved interpreted systems to reason about temporal-epistemic properties in an asynchronous UMAS setting. Similarly to PIS, a PIIS models an unbounded collection of agents where each agent adheres to a different *role*. Each role is associated with a generic *agent template* which specifies the behaviour of each agent of said role. The description of a PIIS consists of the descriptions of a finite number of agent templates and the description of the *environment template*. The environment template expresses the behaviour

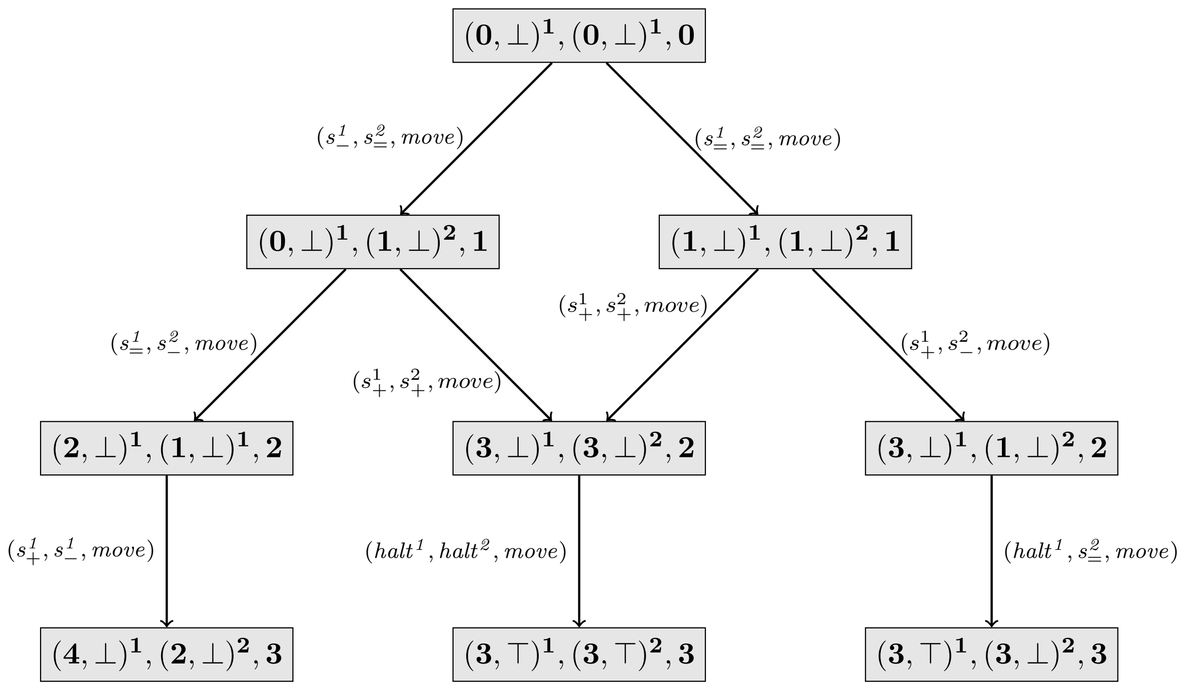

$(\mathbf{0},\bot)^{\mathbf{1}},(\mathbf{0},\bot)^{\mathbf{1}},\mathbf{0}$
$(s^1_-, s^2_=, move)$
$(s^1_=, s^2_=, move)$
$(\mathbf{0},\bot)^{\mathbf{1}},(\mathbf{1},\bot)^{\mathbf{2}},\mathbf{1}$
$(\mathbf{1},\bot)^{\mathbf{1}},(\mathbf{1},\bot)^{\mathbf{2}},\mathbf{1}$
$(s^1_=, s^2_-, move)$
$(s^1_+, s^2_+, move)$
$(s^1_+, s^2_+, move)$
$(s^1_+, s^2_-, move)$
$(\mathbf{2},\bot)^{\mathbf{1}},(\mathbf{1},\bot)^{\mathbf{1}},\mathbf{2}$
$(\mathbf{3},\bot)^{\mathbf{1}},(\mathbf{3},\bot)^{\mathbf{2}},\mathbf{2}$
$(\mathbf{3},\bot)^{\mathbf{1}},(\mathbf{1},\bot)^{\mathbf{2}},\mathbf{2}$
$(s^1_+, s^1_-, move)$
$(halt^1, halt^2, move)$
$(halt^1, s^2_=, move)$
$(\mathbf{4},\bot)^{\mathbf{1}},(\mathbf{2},\bot)^{\mathbf{2}},\mathbf{3}$
$(\mathbf{3},\top)^{\mathbf{1}},(\mathbf{3},\top)^{\mathbf{2}},\mathbf{3}$
$(\mathbf{3},\top)^{\mathbf{1}},(\mathbf{3},\bot)^{\mathbf{2}},\mathbf{3}$

Figure 3.3: Fragment of the concrete system for the autonomous robots with two robots

of the environment under any given number of agents. A parameter for a PIIS is a tuple of natural numbers, one for each role, that determines the actual number of agents in the system. Given a parameter $(n_1, \ldots, n_k)$ for the system, the concrete interleaved interpreted system corresponding to the composition of n_i agents, for each role i, can be constructed.

Assume a set $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ of agent templates. Each template $\mathcal{T}_i = (L_i, \iota_i, Act_i, P_i, t_i)$ is similarly described to a standard agent in interleaved interpreted systems. However, the template $\mathcal{T}_i$ distinguishes between five types of actions: (i) *asynchronous actions*; (ii) *agent-environment actions*; (iii) *role-synchronous actions*; (iv) *global-synchronous actions*; (v) *multi-role actions*. Each type of action is differently shared in a concrete system thus prescribing to a different communication pattern.

1. An asynchronous action from template $\mathcal{T}_i$ is uniquely instantiated for each concrete agent performing role i. Thus, whenever an asynchronous action is performed, exactly one agent is participating in the global transition.

2. An agent-environment action from template $\mathcal{T}_i$, modelling agent-environment communication, is instantiated as an asynchronous action, but each instantiation is shared by

the concrete environment. Hence, whenever an agent-environment action is performed exactly one agent and the environment are participating in the global transition.

3. A role-synchronous action from template $\mathcal{T}_i$, describing multi-agent-environment communication, is shared by all the concrete agents performing role i and the concrete environment. Therefore, whenever a role-synchronous action is performed, all the agents performing role i and the environment are participating in the global transition.

4. A global-synchronous action is shared by all the concrete agents of any role and the concrete environment. Consequently, whenever a global-synchronous action is performed, all the agents and the environment are participating in the global transition.

5. Multi-role actions are designated for pairwise communication between agents performing different roles and the environment. A multi-role action defined for an agent template $\mathcal{T}_i$ is always shared with a second agent template $\mathcal{T}_r$. The set of multi-role actions admitted by template $\mathcal{T}_i$ is the disjoint union $\bigcup_{\mathcal{T}_r} MR_{i,r}$ of the sets of actions shared between template i and every other template $\mathcal{T}_r$; we assume that $MR_{i,r} = MR_{r,i}$. A multi-role action shared between template $\mathcal{T}_i$ and template $\mathcal{T}_r$ is instantiated for each pair of concrete agents performing roles i and r, respectively, and the instantiation is also admitted by the concrete environment. As a result, whenever a multi-role action is performed, the following agents are participating in the global transition: exactly one agent performing role i, exactly one agent performing role r, and the environment.

In section 3.2.1 we exemplify the above synchronisation patterns. First we give the definitions of the agent and environment templates, and the definition of the concrete semantics.

Definition 3.7 (Agent template). *An* agent template $\mathcal{T}_i$ *is an agent* $(L_i, \iota_i, Act_i, P_i, t_i)$ *with a set* $Act_i = A_i \cup AE_i \cup RS_i \cup GS \cup MR_i$ *of actions, where* A_i *is a set* asynchronous *actions,* AE_i *is a set of* agent-environment actions*,* RS_i *is a set of* role-synchronous *actions,* GS *is a set of* global-synchronous actions *that is shared with all other templates, and* $MR_i = \bigcup_{\mathcal{T}_r} MR_{i,r}$ *is a set of multi-role actions. It is assumed that the sets* $A_i, AE_i, RS_i, GS, MR_{i,1}, \ldots, MR_{i,|\mathcal{T}|}$ *are pairwise disjoint.*

The environment template $\mathcal{E}$ is described similarly to an agent template, but for the synchronisation purposes described above, $\mathcal{E}$'s set of actions is the union of the agent templates' sets of agent-environment, role-synchronous, global-synchronous, and multi-role actions.

Definition 3.8 (Environment template). *An* environment template $\mathcal{E} = (L_{\mathcal{E}}, \iota_{\mathcal{E}}, Act_{\mathcal{E}}, P_{\mathcal{E}}, t_{\mathcal{E}})$ *is an agent defined on the set* $Act_{\mathcal{E}} = \bigcup_{\mathcal{T}_i} (AE_i \cup RS_i \cup MR_i) \cup GS$ *of actions.*

A PIIS consists of a finite collection of agent templates and a template environment.

Definition 3.9 (Parameterised interleaved interpreted system). *A* Parameterised Interleaved Interpreted System *is a tuple* $PIIS = (\mathcal{T}, \mathcal{E}, \mathcal{V})$, *where*

- $\mathcal{T} = \{(L_1, \iota_1, Act_1, P_1, t_1), \ldots, (L_k, \iota_k, Act_k, P_k, t_k)\}$ *is a nonempty and finite set of agent templates;*
- $\mathcal{E} = (L_{\mathcal{E}}, \iota_{\mathcal{E}}, Act_{\mathcal{E}}, P_{\mathcal{E}}, t_{\mathcal{E}})$ *is an environment template;*
- $\mathcal{V} = \{V_i : L_i \to \mathcal{P}(AP_i) \colon i \in \mathcal{T}\}$ *is a set of valuation functions, one for each agent template. Each* V_i *is defined on a set of atomic propositions* AP_i*. It is assumed that* $AP_1, \ldots, AP_{|\mathcal{T}|}$ *are pairwise disjoint sets.*

Let $PIIS = (\mathcal{T}, \mathcal{E}, \mathcal{V})$ be a parameterised system of k roles. Let $\overline{n} = (n_1, \ldots, n_k) \in \mathbb{N}^k$ be a concrete value of the system's parameter. Let $\overline{n}.i$ denote the natural number associated with template $\mathcal{T}_i$ in $\overline{n}$. For a k-tuple of natural numbers $\overline{x}$, we write $\overline{x} \leq \overline{n}$ to mean that $\overline{x}.1 \leq \overline{n}.1, \ldots \overline{x}.k \leq \overline{n}.k$. We now describe the $\overline{n}$'st concrete instantiation of a PIIS. The concrete system $PIIS(\overline{n})$ is an interleaved interpreted system that results from the composition of $\overline{n}.i$ instantiations $\{(i,1), \ldots, (i, \overline{n}.i)\}$ of each agent template i and an instantiation of the environment template. Given $\overline{x} \leq \overline{n}$, we write $\mathcal{A}(\overline{x})$ for the set $\mathcal{A}(\overline{x}) = \{(i,j) \colon 1 \leq i \leq k, 1 \leq j \leq \overline{x}.i\}$ of concrete agents. Each concrete agent is instantiated by taking indexed copies of its agent template.

Definition 3.10 (Concrete agent). *Given a PIIS* $PIIS = (\mathcal{T}, \mathcal{E}, \mathcal{V})$ *and a value* $\overline{n} \in \mathbb{N}^k$ *for its parameter, the concrete agent* $(i,j) = \left(L_i^j, \iota_i^j, Act_i^j, P_i^j, t_i^j\right)$ *is defined as follows.*

- $L_i^j = L_i \times \{j\}$, *where* $(\iota_i, j) \in L_i^j$ *is the initial concrete local state;*

- *$Act_i^j = A_i^j \cup AE_i^j \cup RS_i \cup GS \cup MR_i^j$, where:*
  - *$A_i^j = A_i \times \{j\}$;*
  - *$AE_i^j = AE_i \times \{j\}$;*
  - *$MR_i^j = \bigcup_{(r,s)\in\mathcal{A}(\overline{n})} MR_{i,r}^{j,s}$, where $MR_{i,r}^{j,s} = MR_{i,r} \times \{\{(i,j),(r,s)\}\}$ is the set of concrete multi-role actions shared precisely between the concrete agent (i,j) and the concrete agent (r,s) (note that $MR_{i,r}^{j,s} = MR_{r,i}^{s,j}$).*
- *$P_i^j : L_i^j \to \mathcal{P}(Act_i^j)$ is defined for each concrete local state l by*

$$P_i^j(l) = \left\{a\colon a \in Act_i^j \text{ and } a_\mathcal{T} \in P_i(l_\mathcal{T})\right\},$$

  where $a_\mathcal{T}$ ($l_\mathcal{T}$, respectively) denotes the corresponding template action (state, respectively) from which a (l, respectively) has been instantiated;
- *$t_i^j : L_i^j \times Act_i^j \to L_i^j$ is such that*

$$t_i^j(l,a) = l' \text{ iff } t_i(l_\mathcal{T}, a_\mathcal{T}) = l'_\mathcal{T}.$$

So, the local states of a concrete agent are made of the template local states indexed by the name of the agent in question. The agent inherits from its template the actions, the protocol and the transition function. The concrete environment is similarly obtained by instantiating each action shared with the agent templates.

Definition 3.11 (Concrete environment)**.** *Given a PIIS $PIIS = (\mathcal{T}, \mathcal{E}, \mathcal{V})$ and a value $\overline{n} \in \mathbb{N}^k$ for its parameter, the concrete environment $E = (L_E, \iota_E, Act_E, P_E, t_E)$ is defined as follows.*

- *$L_E = L_\mathcal{E}$ and $\iota_E = \iota_\mathcal{E}$;*
- *$Act_E = \bigcup_{(i,j)\in\mathcal{A}(\overline{n})} \left(AE_i^j \cup MR_i^j\right) \cup \bigcup_{i\in\mathcal{T}} RS_i \cup GS$;*
- *$P_E : L_E \to \mathcal{P}(Act_E)$ is defined for each local state l by*

$$P_E(l) = \{a\colon a \in Act_E \text{ and } a_\mathcal{T} \in P_\mathcal{E}(l_\mathcal{T})\};$$

- $t_E : L_E \times Act_E \to L_E$ *is such that*

$$t_E(l, a) = l' \textit{ iff } t_{\mathcal{E}}(l_\tau, a_\tau) = l'_\tau.$$

Finally, a PIIS's instantiation, and the concrete semantics we consider for asynchronous UMAS, is the interleaved interpreted system composed of the concrete agents and the concrete environment.

Definition 3.12 (Concrete system). *Given a PIIS $PIIS = (\mathcal{T}, \mathcal{E}, \mathcal{V})$ and a value $\overline{n} \in \mathbb{N}^k$ for its parameter, the* concrete system $PIIS(\overline{n})$ *is the interleaved interpreted system*

$$PIIS(\overline{n}) = \left((L_i^j, \iota_i^j, Act_i^j, P_i^j, t_i^j)_{(i,j)\mathcal{A}(\overline{n})}, (L_E, \iota_E, Act_E, P_E, t_E), \iota, V \right),$$

where:

- $\iota = ((\iota_1, 1), \dots, (\iota_k, \overline{n}.k), \iota_E)$ *is the unique initial global state;*
- $V : S \to \mathcal{P}(AP)$ *is the concrete valuation defined on the set $S = L_1^1 \times \dots \times L_k^{\overline{n}.k} \times L_E$ of possible global states, and on the set $AP = (AP_1 \times \{1, \dots, \overline{n}.1\}) \cup \dots \cup (AP_k \times \{1, \dots, \overline{n}.k\})$ of atomic propositions as follows:*

$$\textit{for } p \in AP_i \textit{ and } 1 \leq j \leq \overline{n}.i, (p, j) \in V(g) \textit{ iff } p \in V_i(l),$$

where l is the template local state of agent (i, j) in g.

For each concrete system $PIIS(\overline{n})$ we can associate a temporal-epistemic model $\mathcal{S}_{PIIS(\overline{n})} = \left(G(\overline{n}), \iota(\overline{n}), R(\overline{n}), (\mathcal{K}_i^j)_{(i,j)\in\mathcal{A}(\overline{n})}, V(\overline{n})\right)$ as standard (see section 2.1.2). When $PIIS(\overline{n})$ is clear from the context we simply write $\mathcal{S}(\overline{n})$ for $\mathcal{S}_{PIIS(\overline{n})}$. Given a global state g, we write $ls_i^j(g)$ for the local state of agent (i, j) in g. The template local state of agent (i, j) in g is denoted by $tls_i^j(g)$.

In compliance with the interleaved semantics, we can distinguish five types of transitions on a concrete system. In particular, a global transition from a state g can only happen in the following cases: (i) a concrete asynchronous A_i^j action is enabled for agent (i, j) performing

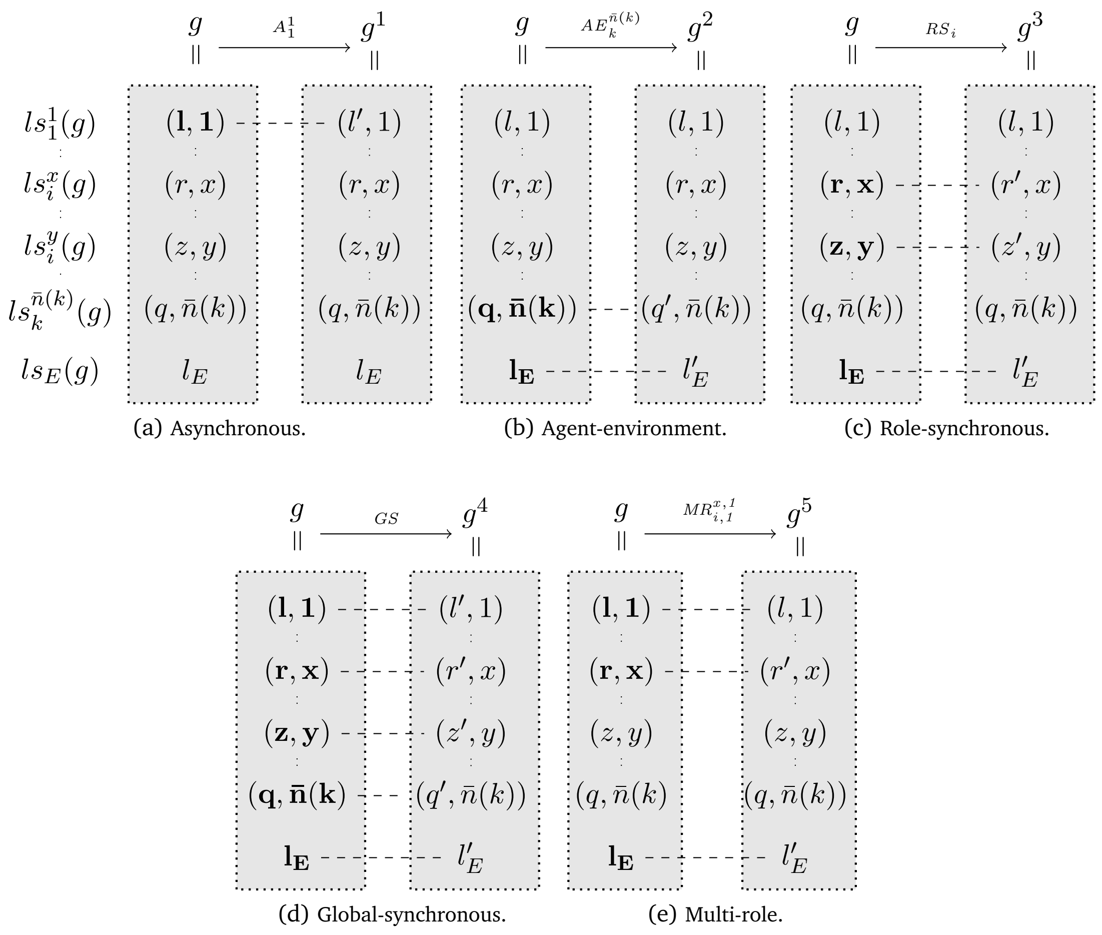
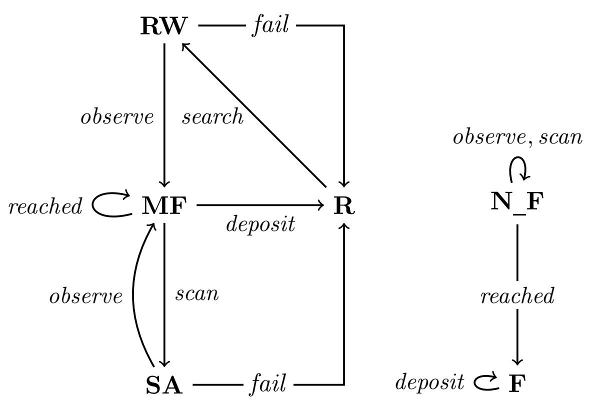


Figure 3.4: Examples of the five types of transitions possible in a concrete evolution from a global state.

role i at g; (ii) a concrete agent-environment AE_i^j action is enabled for the environment and for agent (i,j) performing role i at g; (iii) a concrete role-synchronous RS_i action is enabled for the environment and for all the agents performing role i at g; (iv) a concrete global-synchronous GS action is enabled for the environment and for all the agents at g; (v) a concrete multi-role $MR_{i,r}^{j,q}$ action is enabled for the environment, for agent (i,j) performing role i, and for agent (r,q) performing role r at g. These transitions are depicted in Figure 3.4: (a) asynchronous for agent $(1,1)$; (b) agent-environment for agent $(k,\bar{n}(k))$ and the environment; (c) role-synchronous for all the agents from template $\mathcal{T}_i$ and the environment; (d) global-synchronous for all the agents and the environment; (e) multi-role for agents (i,x), $(1,1)$ and the environment. Symbols in bold indicate the components of a global state on which the enabling of each action depends. Dashed lines from a global state denote the components in the state that are updated upon the corresponding global transition.



(a) Template robot. (b) Template food source.

Figure 3.5: The parameterised interleaved interpreted system for the robot foraging scenario.

3.2.1 Examples

We exemplify the technical notions introduced above on three examples: the Train-Gate-Controller model [HW02b], a robot foraging scenario [Liu07], an autonomous robot example [FHMV95]. The Train-Gate-Controller illustrates the agent-environment and the global-synchronous communication patterns. The robot-foraging scenario gives an intuitive example of multi-role synchronisations. We discuss role-synchronous communication in the context of the autonomous robot example. We here focus on the semantic modelling. We will later discuss specifications in section 3.3 and verification methodologies in chapters 5 and 6.

Robot Foraging Scenario

In the following we describe an untimed version of the robot foraging scenario (RFS) introduced in [Liu07]. The RFS includes an arbitrary number of robots initially resting in a nest before undertaking a campaign in search for food by means of a random walk. Upon observing a food source, a robot tries to reach for it. If it succeeds, then (i) it collects and deposits the food in the nest; (ii) it makes the location of the food known so that all other robots can find it. Otherwise, if it fails to reach the food source, it then scans the area to locate the source again, or locate a new source. If the scan is successful, then the robot attempts to reach the food source. Otherwise, if the scan is not successful (under a timeout), then the robot returns to its nest.

We can encode the scenario as a PIIS $\mathcal{S}_{RFS}$ composed of an agent template TR representing the robots and an agent template TFS representing the food sources. The template robot is depicted in Figure 3.5a. In the figure R stands for "Resting", RW for "Random Walk", MF for "Move to Food", and SA for "Scan Area". TR is initially in state R representing that the robot is resting in its nest. The states RW, MF, SA represent that the robot is performing a random walk, the robot is moving to the food, and the robot is scanning the area, respectively. The template food source is given by Figure 3.5b. In the figure N_F stands for "Not Found" and F stands for "Found". TFS is initially in the state N_F representing that the food source has not been found. The state F represents that the food source is found.

We now describe the global transitions induced by the templates. As discussed in the previous section, a multi-role action is always admitted in the repertoire of actions of precisely two agent templates. The action is instantiated for each pair of agents instantiated from the two templates. In a global transition induced by a multi-role action only the agents for which the action is instantiated and the environment are participating in the transition.

- $search$. This an asynchronous action that is defined for the template robot. It is enabled at state R and it represents a robot moving out of its nest to search for food. A global transition by means of the $search$ action results the robot performing the action to move to state RW.

- $fail$. This is also an asynchronous action that is enabled at states RW and SA of the template robot. The action represents a robot failing to locate a food source when performing a random walk and when scanning the area, respectively. A global transition via the $fail$ action results the robot performing the action to move to state R.

- $observe$. This is a multi-role action shared between the two templates. Assume the instantiation $(observe, \{(TR, i), (TFS, j)\})$ of the action for robot i and food source j. A concrete transition via the $(observe, \{(TR, i), (TFS, j)\})$ action is only enabled if the robot i is either in state RW or in state SA, and the food source j is in state N_F. Intuitively the robot can observe the food source if the latter has not already been found. The action causes the robot i to change its state to MF.

- $\mathit{reached}$. This is also a multi-role action that is shared between the two templates. Following the concrete transition described above, a concrete transition by means of the $(\mathit{reached}, \{(\mathit{TR}, i), (\mathit{TFS}, j)\})$ action is enabled. This transition causes the food source j to change its state to F, thus modelling that robot i has succeeded in reaching the food source j.

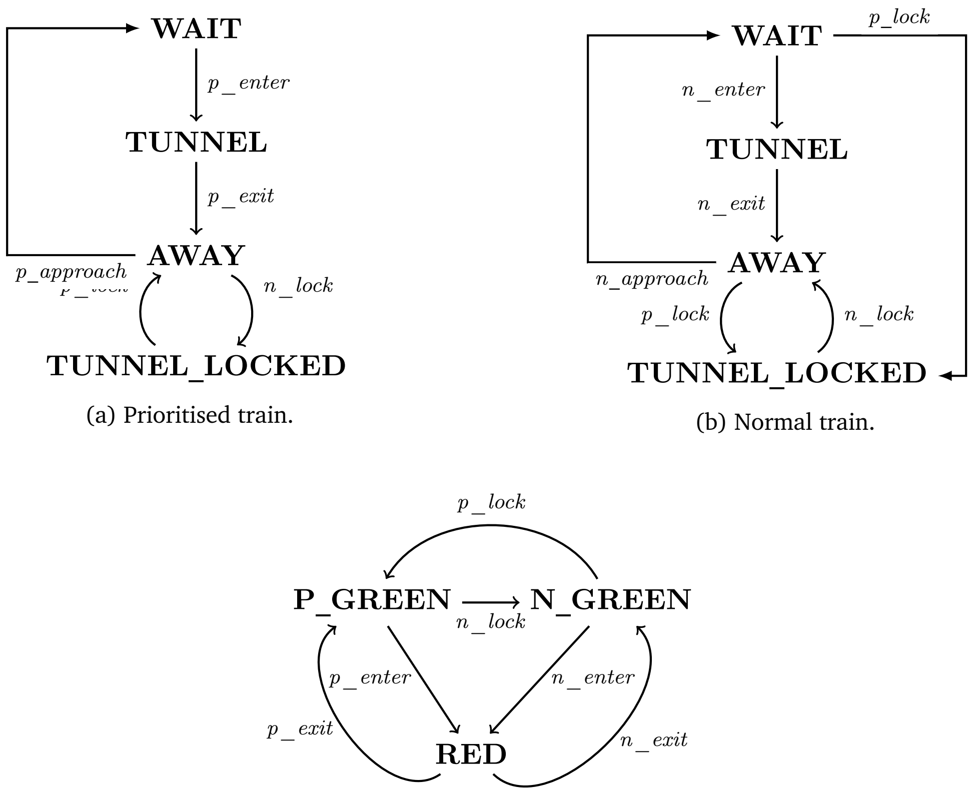
- $\mathit{deposit}$. The above transition enables the multi-role action $(\mathit{deposit}, \{(\mathit{TR}, i), (\mathit{TFS}, j)\})$. A transition via this action causes the robot i to move to state R.

- scan. Finally, scan is also a multi-role action. Intuitively, robot i may fail to reach the food source j (i.e., the $(\mathit{reached}, \{(\mathit{TR}, i), (\mathit{TFS}, j)\})$ action is not performed). In this case the $(\mathit{scan}, \{(\mathit{TR}, i), (\mathit{TFS}, j)\})$ action is enabled. Upon this transition, the robot i updates its state to SA.

The Train-Gate-Controller

In section 2.1.2 we defined the IIS of the Train-Gate-Controller (TGC) composed of a controller and two trains. We now give the PIIS model of a parameterised version of the TGC. We extend the original description to include an arbitrary number of two types of trains: *prioritised trains* and *normal trains*. A prioritised train can enter the tunnel at any given time, assuming there is no other train in the tunnel, whereas a normal train can only enter the tunnel when there is no other train waiting to enter the tunnel. To accomplish this, the traffic lights include two shades of the green colour: prioritised green and normal green. Prioritised green is used by the controller to serve prioritised trains, whereas normal green is used by the controller to serve normal trains.

The scenario can be encoded as a PIIS composed of an agent template representing prioritised trains (Figure 3.6a), an agent template representing normal trains (Figure 3.6b), and an environment template representing the controller (Figure 3.6c). A prioritised train is initially in state WAIT, the controller is initially in state $\mathit{P\_GREEN}$, and a normal train is initially in state $\mathit{TUNNEL\_LOCKED}$. Therefore, prioritised trains are initially waiting to enter the tunnel, normal trains are initially locked from entering the tunnel, and the controller initially serves only prioritised trains. The actions $\mathit{p\_enter}$ and $\mathit{p\_exit}$ are agent-environment actions



(a) Prioritised train.

(b) Normal train.

(c) Controller.

Figure 3.6: The parameterised interleaved interpreted system for the Train-Gate-Controller.

modelling the prioritised trains entering and exiting the tunnel. Similarly, the actions n_enter and n_exit are agent-environment actions enabling the normal trains to enter and exit the tunnel. The action n_lock is a global-synchronous action; it represents the normal trains taking the lock on the tunnel. Also, the action p_lock is a global-synchronous action; it models the prioritised trains taking the lock on tunnel. Finally, the actions $p\_approach, n\_appoach$ are asynchronous actions.

The templates induce the following agent-environment and global-synchronous concrete transitions:

- p_enter, n_enter. In addition to the agent performing the action entering the tunnel, the environment participates in the global transition. This causes the environment to change its state to RED, thereby disallowing other trains to enter the tunnel.
- p_exit, n_exit. The environment synchronises with the agent that is currently in the tunnel via the p_exit and n_exit actions. The synchronisation causes the environment to change its state to P_GREEN, if the agent is a prioritised train, or to N_GREEN, if the agent is a normal train. Following this, other trains are allowed to enter the tunnel.
- n_lock. This action is only enabled if: (i) the environment is in state P_GREEN; (ii) there is no train in the tunnel; (iii) all prioritised trains are in state $AWAY$. A concrete global-transition via the n_lock action causes the environment to update its state to N_GREEN. Thus, the transition frees the tunnel to serve normal trains whenever there are no prioritised trains waiting to be served.
- p_lock. This action is only enabled if: (i) the environment is in state N_GREEN; (ii) there is no train in the tunnel. Upon performing this action the environment moves to state P_GREEN. Therefore, the transition locks the tunnel to serve prioritised trains; this can happen irrespective of whether there are normal trains waiting to be served.

The above transitions are depicted in Figure 3.7 for a fragment of the concrete system with two prioritised trains and two normal trains. Each global state in the figure is a 5-tuple representing, from left to right, the local state of the first prioritised train, the local state of the second prioritised train, the local state of the controller, the local state of the first normal train, and the local state of the second normal train. In the figure W stands for $WAIT$, PG for P_GREEN, L for $TUNNEL\_LOCKED$, T for $TUNNEL$, R for RED, A for $AWAY$, and NG for N_GREEN.

Autonomous robot with a unique shared sensor

Section 3.1.1 described a generalisation of the autonomous robot example from [FHMV95] in which an arbitrary number of robots run synchronously along the track. We here assume that the robots have access to a unique shared sensor. To illustrate the role-synchronous actions, we introduce a second type of robots, identical to the description of the first type, but with no access to a sensor. We refer to the two types of robots as type 1 robots and type 2 robots, respectively. Type 2 robots halt after receiving a *halting event* from type 1 robots. The event is signaled after the type 1 robots have halted.

We encode the scenario as a PIIS $\mathcal{S}_{AR}$ composed of an agent template $TR1$ representing robots with access to a sensor, an agent template $TR2$ representing robots with no access to a sensor, and an environment template $\mathcal{E}$ for synchronisation purposes. The encoding assumes a finite track with 8 distinct locations.

$TR1$ is given by Figure 3.8a. A template state represents the position of the robot, its sensor reading, and whether it has halted or not, respectively. $TR2$ is depicted in Figure 3.8b. A template state represents the position of the robot and whether it has halted or not, respectively. Finally, $\mathcal{E}$ is defined by Figure 3.8c. A template state represents the position of the robots and whether or not the type 1 robots have halted. The templates induce the following role-synchronous and global-synchronous transitions:

- $move^+, move^=, move^-$. These are global-synchronous actions. A concrete transition via these actions causes all the robots to move one step forwards. Additionally, type 1 robots change their sensor reading to be either the correct reading ($move^=$), the correct reading plus 1 ($move^+$), or the correct reading minus 1 ($move^-$).
- $halt$. The role-synchronous action $halt$ is enabled at any state in which the sensor reading

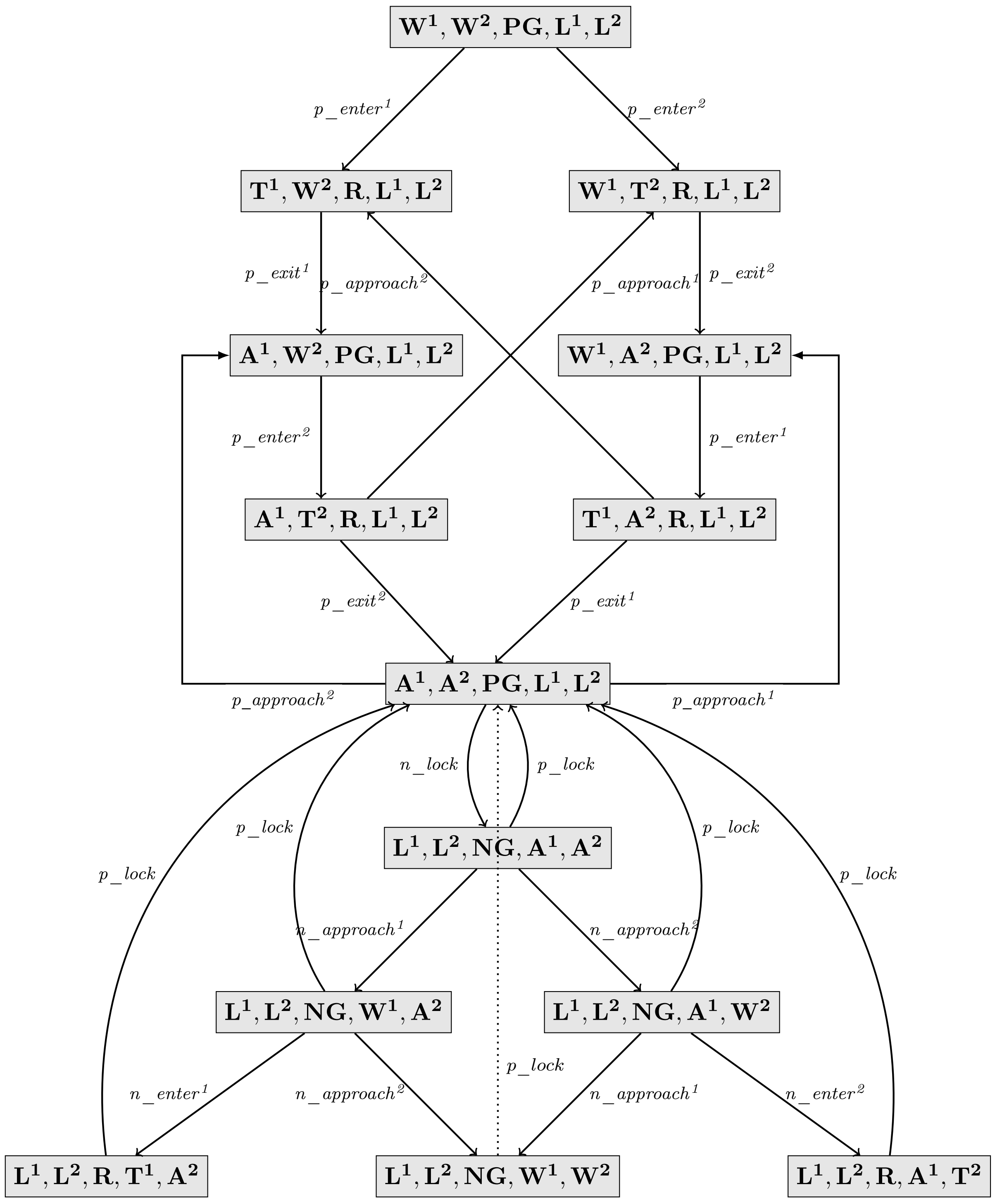


Figure 3.7: Fragment of the concrete system for the train-gate-controller with two prioritised trains and two normal trains.

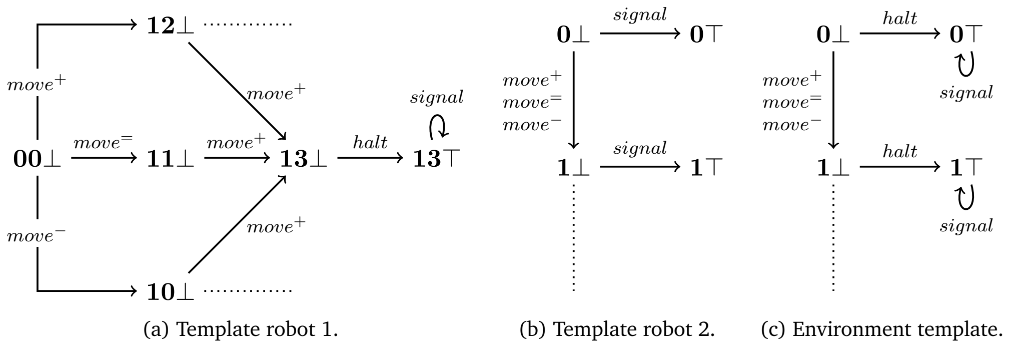


Figure 3.8: The parameterised interleaved interpreted system of the autonomous robot.

of type 1 robots is greater than or equal to 3. Type 1 robots halt upon this transition and the environment stores in its state the fact that they have halted.

- $signal$. Following the above transition, a concrete transition via the global-synchronous action $signal$ is enabled. The transition causes the type 2 robots to halt.

3.3 Specifications for unbounded multiagent systems

To establish the correctness of a UMAS irrespectively of the number of agents present, we express properties that reflect its parameterised nature. In other words, we are interested in expressing properties irrespectively of the number of agents in the system.

Such properties are not easily expressible by means of propositional temporal epistemic logics. Indeed, suppose that we want to express the property "for every robot i, whenever i is halted, it knows that in the next two time steps every other robot is halted" for the autonomous robot scenario (section 3.1.1). This property encodes all distinct pairs of robots. Therefore, to express the property for a concrete system composed of n robots we need to construct a formula composed of of $2!\binom{n}{2}$ conjuncts. Instead we would like to express properties that are independent of the number of agents in the system, as if we were able to quantify over the agents.

To overcome these shortcomings we consider an indexed variant of the temporal-epistemic logic CTL*K. This logic introduces indexed atomic propositions and indexed epistemic modalities.

3.3.1 Syntax of indexed CTL*K

Let $\mathcal{S}$ denote either a PIS $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{I}, \mathcal{V})$ or a PIIS $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V})$ of $|\mathcal{T}| = k$ roles and assume that each valuation function $V_i \in \mathcal{V}$ is defined on the set AP_i of atomic propositions, where $AP_1, \ldots, AP_k$ are pairwise disjoint sets. Let $VAR = \{VAR_i \colon 1 \leq i \leq k\}$ be a set of pairwise disjoint sets of variable symbols. Each set VAR_i is associated with agent template $\mathcal{T}_i$. We now define indexed CTL*K.

Definition 3.13. *(Syntax of indexed CTL*K) State and path formulae of indexed CTL*K over a set of templates $\mathcal{T}$, the sets $AP_1, \ldots, AP_{|\mathcal{T}|}$ of atomic propositions, and the sets $VAR_i, \ldots, VAR_{|\mathcal{T}|}$ of variable symbols are inductively defined as follows.*

S1. *every atomic proposition (p, v) ($p \in AP_i, \mathcal{T}_i \in \mathcal{T}, v \in VAR_i$) is a state formula;*

S2. *if φ and ψ are state formulas, then $\neg\varphi$, $\varphi \vee \psi$ and $K_i^v \phi$ ($\mathcal{T}_i \in \mathcal{T}, v \in VAR_i$) are state formulas;*

S3. *if φ is a path formula, then $E(\varphi)$ is a state formula;*

P1. *any state formula φ is also a path formula;*

P2. *if φ and ψ are path formulas, then $\neg\varphi$ and $\varphi \vee \psi$ are path formulas;*

P3. *if φ and ψ are path formulas, then so are $X\varphi$ and $\varphi U \psi$.*

Therefore, the state and path formulae of indexed CTL*K are defined as the state and path formulae of CTL*K, but with each atomic proposition $p \in AP_i$ and epistemic modality K_i indexed by a variable $v \in VAR_i$. The domain of a variable $v \in VAR_i$ appearing in a formula ϕ is defined by the concrete system on which ϕ is evaluated: if ϕ is evaluated on $\mathcal{S}(\overline{n})$, then the potential set of values for v is $\{1, \ldots, \overline{n}.i\}$. That is, the domain of v is the set of identities for the concrete agents performing role i. We write $\phi(\overline{v})$, where $\overline{v} = (V_1, \ldots, V_k)$ is a k-tuple of sets of variables, to indicate that each of the variables in $V_i \subseteq VAR_i$ appears in an atomic proposition or epistemic modality in ϕ. We say that $\phi(\overline{v})$ is an $\overline{m}$-indexed CTL*K formula, where $\overline{m}$ is k-tuple of natural numbers, if $|V_i| = \overline{m}.i$, for all $1 \leq i \leq k$. The properties we use in the verification of UMAS are constructed in compliance with the following formula schema:

$$\forall v_1 \ldots \forall v_{\overline{m}.1} \ldots \forall u_1 \ldots \forall u_{\overline{m}.k} \left(\left(\bigwedge_{1 \leq i \leq \overline{m}.1, 1 \leq j \leq \overline{m}.1, i \neq j} \neg(v_i = v_j) \wedge \ldots \wedge \right.\right.$$
$$\left.\left. \bigwedge_{1 \leq i \leq \overline{m}.k, 1 \leq j \leq \overline{m}.k, i \neq j} \neg(u_i = u_j) \right) \rightarrow \phi(\overline{v}) \right)$$

where $\{v_1, \ldots, v_{\overline{m}.1}\} = V_1, \ldots, \{u_1, \ldots, u_{\overline{m}.k}\} = V_k$. We denote such a formula by $\forall_{\overline{v}} \phi(\overline{v})$.

Using the above schema we may now express properties independently of the number of agents as the following examples illustrate.

Example 3.1. *In section 3.2.1 we encoded the robot foraging scenario as a PIIS composed of an agent template TR representing the robots and an agent template TFS representing the food sources. TFS was built from two states expressing whether or not a food source has been found by a robot. Having modelled the scenario as a PIIS we are interested in checking the property: "whenever a food source is found, every robot knows that the source is found". This property can be expressed by the following $(1,1)$-indexed formula:*

$$\phi_{RFS} = \forall_{(\{u\},\{x\})} AG\left((f, x) \rightarrow K^u_{TR}(f, x)\right)$$

where u is a variable of TR, x is a variable of TFS, and the atomic proposition f holds in the template state in which the template food source is "found".

Example 3.2. *In section 3.2.1 we encoded the train-gate-controller as a PIIS composed of an agent template PT representing prioritised trains, an agent template representing NT representing normal trains, and a template environment representing the controller. A commonly used benchmark concerns assessing the correctness of the train-gate-controller against the property "whenever a train is in the tunnel, it knows that no other train is in the tunnel at the same time". This property can be expressed in indexed ACTL*$K \backslash X$ by the following $(2,2)$-indexed formula:*

$$\phi_{TGC} = \forall_{(\{u,v\},\{x,y\})} AG\left(\left((pt, u) \rightarrow K^u_{PT}\left(\neg(pt, v) \wedge \neg(nt, x)\right)\right)\right.$$
$$\left.\wedge \left((nt, x) \rightarrow K^x_{NT}\left(\neg(nt, y) \wedge \neg(pt, u)\right)\right)\right)$$

where u, v are variables of PT, x, y are variables of NT, the atomic proposition pt holds in the *template states in which the template prioritised train is in the tunnel, and the atomic proposition nt holds in the template states in which the template normal train is in the tunnel.*

Example 3.3. *In section 3.2.1 we encoded the autonomous robot example as a PIIS composed of an agent template $TR1$ representing robots with access to a sensor, an agent template $TR2$ representing robots with no access to a sensor, and an environment template representing the environment moving the robots forward along the track. According to the scenario, the goal of the robots is to halt in the goal region $\{2, 3, 4\}$ of the track. A solution to the problem in the single-robot case is for the robot to do nothing while the value of its sensor is less than 3 and to halt once the value of its sensor is greater than or equal to 3. We are interested to show the correctness of the autonomous robot solution in the unbounded case, i.e, whenever the robots halt, they know that they are in the goal region. This is expressed by the following $(1, 1)$-indexed formula:*

$$\phi_{AR} = \forall_{(\{v\},\{x\})} AG((h\_1, v) \rightarrow K^{v}_{TR1}((gr\_1, v)) \wedge (h\_2, u) \rightarrow K^{u}_{TR2}((gr\_2, u)))$$

where v is a variable of $TR1$, u is a variable of $TR2$, the atomic proposition gr_1 (gr_2, respectively) holds in the template states where the value of the position component of template robot 1 (template robot 2, respectively) is in $\{2, 3, 4\}$, and the atomic proposition h_1 (h_2, respectively) holds in the template states where the template robot 1 (template robot 2, respectively) has halted.

Several subsets of indexed CTL*K can be defined in a similar manner to Definition 2.8. Chapter 4 studies the verification of PIS against specifications expressed in indexed ACTLK. Chapter 5 analyses the verification of PIIS against specifications expressed in indexed ACTL*K$\backslash X$.

3.3.2 Satisfaction of indexed CTL*K

When evaluated on a concrete system, an $\overline{m}$-indexed formula $\forall_{\overline{v}}\phi(\overline{v})$ denotes a specific CTL*K formula corresponding to the conjunction of all *ground instantiations* of $\forall_{\overline{v}}\phi(\overline{v})$. Given a concrete system $\mathcal{S}(\overline{n})$ with $\overline{n}.1 \geq \overline{m}.1, \ldots, \overline{n}.k \geq \overline{m}.k$, and a set of injective functions

$$\{\xi_i : V_i \rightarrow \{1, \ldots, \overline{n}.i\} : 1 \leq i \leq k\},$$

a ground instantiation of $\forall_{\overline{v}}\phi(\overline{v})$ is the CTL*K formula obtained from $\phi(\overline{v})$ by assigning to each variable $v \in V_i$ the value $\xi_i(v)$, for all $1 \leq i \leq k$.

Definition 3.14 (Satisfaction of indexed CTL*K). *Given an $\overline{m}$-indexed CTL*K formula $\forall_{\overline{v}}\phi(\overline{v})$ and a model $\mathcal{S}(\overline{n})$ of k roles, the satisfaction relation $\models$ is defined as follows:*

$$\mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v}) \textit{ iff } \mathcal{S}(\overline{n}) \models \bigwedge_{\overline{\xi}} \phi[\overline{\xi}],$$

where $\bigwedge_{\overline{\xi}}$ denotes the conjunction of all ground instantiations of $\forall_{\overline{v}}\phi(\overline{v})$, and $\overline{n}.1 \geq \overline{m}.1, \ldots, \overline{n}.k \geq \overline{m}.k$.

For example, when evaluated on a concrete system with 2 robots and 2 food sources the formula

$$\phi_{RFS} = \forall_{(\{u\},\{x\})} AG\left((f,x) \rightarrow K^u_{TR}(f,x)\right)$$

is a shorthand for the ACTL*K$\backslash X$ formula

$$AG((f,1) \rightarrow K^1_{TR}(f,1)) \wedge AG((f,1) \rightarrow K^2_{TR}(f,1)) \wedge$$
$$AG((f,2) \rightarrow K^1_{TR}(f,2)) \wedge AG((f,2) \rightarrow K^2_{TR}(f,2))$$

Note that since an $\overline{m}$-indexed formula refers to $\overline{m}.i$-tuples of distinct agents, for each role i, the satisfaction relation above is well defined only if $\overline{n}.1 \geq \overline{m}.1, \ldots, \overline{n}.k \geq \overline{m}.l$. We write $\overline{n} \geq \overline{m}$ to express this.

3.3.3 Symmetry reduction

Consider an $\overline{m}$-indexed formula $\forall_{\overline{v}}\phi(\overline{v})$. As noted above, the evaluation of the formula on a concrete system $\mathcal{S}(\overline{n})$ corresponds to the evaluation of the conjunction of all its ground instantiations. But the conjuncts are identical up to re-indexing of the agents' indices. Thus, the symmetric nature of $\forall_{\overline{v}}\phi(\overline{v})$ suggests that its evaluation on a concrete system is equivalent to checking only one of its ground instantiations. The following Lemmas, adapted from [ES96], show precisely this by taking (for simplicity) the aforementioned ground formula to be the *trivial instantiation*. The trivial instantiation of $\forall_{\overline{v}}\phi(\overline{v})$, written $\phi[trivial]$, is the ground formula resulting from assigning the values $\{1,\ldots,\overline{m}(i)\}$ to the variables appearing in each set of variables V_i; i.e; for an assignment ξ_i defined as $\xi_i(v_1) = 1,\ldots,\xi_i(|V_i|) = |V_i|$ for each agent template i, $\phi[trivial]$ is the ACTL*K$\backslash X$ formula obtained from $\forall_{\overline{v}}\phi(\overline{v})$ by assigning to each variable $v \in V_i$ the value $\xi_i(v)$.

Example 3.4. *The following formulae are the trivial instantiations of ϕ_{RFS}, ϕ_{TGC} and ϕ_{AR}, respectively.*

$$\phi_{RFC}[trivial] = AG((f,1) \rightarrow K^1_{TR}(f,1))$$

$$\phi_{TGC}[trivial] = AG(((pt,1) \rightarrow K^1_{PT}(\neg(pt,2) \wedge \neg(nt,1))) \wedge ((nt,1) \rightarrow K^1_{NT}(\neg(nt,2) \wedge \neg(pt,1))))$$

$$\phi_{AR}[trivial] = AG((h\_1,1) \rightarrow K^1_{TR1}((gr\_1,1)) \wedge (h\_2,1) \rightarrow K^1_{TR2}((gr\_2,1)))$$

Lemma 3.1 (Symmetry reduction for PIIS). *Let $\mathcal{S}$ be a PIIS composed of k roles, CTL*K be an $\overline{m}$-indexed formula $\forall_{\overline{v}}\phi(\overline{v})$, and $\overline{n} \geq \overline{m}$. Then, $\mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v})$ iff $\mathcal{S}(\overline{n}) \models \phi[trivial]$.*

Proof. For the left to right direction, suppose that $\mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v})$. As $\forall_{\overline{v}}\phi(\overline{v})$ expresses the conjunction of all its instantiations we have that $\mathcal{S}(\overline{n}) \models \phi[\overline{\xi}]$ for any k-tuple of assignments $\overline{\xi}$. Therefore, $\mathcal{S}(\overline{n}) \models \phi[trivial]$.

For the right to left direction, suppose that $\mathcal{S}(\overline{n}) \models \phi[trivial]$. So, for each $1 \leq i \leq k$, the variables $v_1,\ldots,v_{\overline{m}.i}$ in $\overline{v}.i$ are mapped into the concrete agents $(i,1),\ldots,(i,\overline{m}.i)$, respectively. Let $\overline{\xi} = (\xi_1,\ldots,\xi_k)$ be an arbitrary assignment.

For each $1 \leq i \leq k$, consider ζ_i to be either a bijective mapping from natural numbers to natural numbers, or a bijective mapping from global states to global states, or a bijective mapping from actions to actions, or a bijective mapping from CTL*K formulae to CTL*K formulae, the exact mapping depending on the context.

In the first case, define $\zeta_i : \{1,\ldots,\overline{n}.i\} \rightarrow \{1,\ldots,\overline{n}.i\}$ to be an arbitrary mapping that satisfies $\zeta_i(j) = \xi_i(v_j)$ for each $1 \leq j \leq \overline{m}.i$.

For the second case, recall that for a global state g, $ls^j_i(g)$ returns the concrete local state of agent (i,j) in g, and $tls_i^j(g)$ returns the template local state of agent (i,j) in g. Then define $\zeta_i : G(\overline{n}) \rightarrow G(\overline{n})$ by $\zeta_i(g) = g'$ iff $tls_i^j(g') = tls_i^{\zeta_i^{-1}(j)}(g)$, for each $1 \leq j \leq \overline{n}.i$, and $ls_E(g') = ls_E(g)$. For example, if $k = 1, \overline{n}.1 = 3$, and $\zeta_i : \{1,2,3\} \rightarrow \{1,2,3\}$ is given by $\{(1,1) \mapsto (1,3), (1,2) \mapsto (1,1), (1,3) \mapsto (1,2)\}$, then the state $((s,1),(t,2),(u,3),l_E)$ is mapped to the state $((t,1),(u,2),(s,3),l_E))$.

Next, define $\zeta_i : \left(Act_i^1 \cup \ldots \cup Act_i^{\overline{n}.i}\right) \rightarrow \left(Act_i^1 \cup \ldots \cup Act_i^{\overline{n}.i}\right)$ as follows:

- if $(a,j) \in A_i^j \cup AE_i^j$ is an asynchronous action or an agent-environment action of agent (i,j), then $\zeta_i((a,j)) \rightarrow (a, \zeta_i(j))$;
- if $a \in RS_i \cup GS$ is a role-synchronous or a global-synchronous action of agent (i,j), then $\zeta_i(a) \rightarrow a$;
- if $(a, \{(i,j),(q,r)\}) \in MR_{i,q}$ is a multi-role action of agent (i,j) that is shared with an agent (q,r), then $\zeta_i\left((a,\{(i,j),(q,r)\})\right) \rightarrow (a, \{(i,\zeta_i(j)),(q,\zeta_q(r))\})$;

In the latter case, define ζ_i to map CTL*K formulae to CTL*K formulae by $\zeta_i(\phi) = \phi'$ iff ϕ' is the formula obtained from ϕ by replacing each atomic proposition $(p,j) \in AP_i \times \{1,\ldots,\overline{n}.i\}$ with $(p,\zeta_i(j))$, and each epistemic modality K_i^j with $K_i^{\zeta_i(j)}$.

Now consider $\zeta = \zeta_1 \circ \ldots \circ \zeta_k$ to be the composition of $\zeta_1,\ldots,\zeta_k$. Then, in compliance with the interleaved semantics, we have that $g \rightarrow_a g'$ iff $\zeta(g) \rightarrow_{\zeta(a)} \zeta(g')$ and $\mathcal{K}_i^j(g,g')$ iff $\mathcal{K}_i^{\zeta_i(j)}(\zeta(g),\zeta(g'))$. Therefore, by assumption we obtain $(\mathcal{S}(\overline{n}),\zeta(\iota(\overline{n}))) \models \zeta(\phi[trivial])$. Hence, $(\mathcal{S}(\overline{n}),\zeta(\iota(\overline{n}))) \models \phi[\overline{\xi}]$. But, as all concrete agents share a unique template local state in $\iota(\overline{n})$, it follows that $\zeta(\iota(\overline{n})) = \iota(\overline{n})$. Consequently, $(\mathcal{S}(\overline{n}),\iota(\overline{n})) \models \phi[\overline{\xi}]$. As $\overline{\xi}$ was arbitrary, the latter concludes $\mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v})$. □

Lemma 3.2 (Symmetry reduction for PIS). *Let $\mathcal{S}$ be a PIS composed of k roles, CTL*K be an $\overline{m}$-indexed formula $\forall_{\overline{v}}\phi(\overline{v})$, and $\overline{n} \geq \overline{m}$. Then, $\mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v})$ iff $\mathcal{S}(\overline{n}) \models \phi[trivial]$.*

Proof sketch. The proof is along the same lines as the proof of Lemma 3.1. □

3.4 Equivalences

Section 2.3 discussed notions of simulations between temporal-epistemic models that preserve the satisfaction of temporal-epistemic formulae. This section adapts these simulation ideas in the context of concrete systems generated by parameterised (interleaved) interpreted systems. By means of the results here discussed, the truth of a specification on an arbitrarily big concrete system can be assessed by checking it on a smaller, similar system. In the case of PIIS, a specification is given in indexed ACTL*K$\backslash X$. In the case of PIS, a specification is given in indexed ACTLK.

3.4.1 Equivalences preserving indexed ACTL*K$\backslash X$

We define a variant of the notion of stuttering simulation between concrete systems. We show that similar systems preserve indexed ACTL*K$\backslash X$ specifications.

Consider a PIIS $\mathcal{S}$ of k roles. Let $\mathcal{S}(\overline{n}) = \Big(G(\overline{n}), \iota(\overline{n}), R(\overline{n}), (\mathcal{K}_i^j)_{(i,j)\in\mathcal{A}(\overline{n})}, V(\overline{n})\Big)$ and $\mathcal{S}(\overline{n}') = \Big(G(\overline{n}'), \iota(\overline{n}'), R(\overline{n}'), (\mathcal{K}'^j_i)_{(i,j)\in\mathcal{A}(\overline{n}')}, V(\overline{n}')\Big)$ be two concrete instantiations of $\mathcal{S}$. By Lemma 3.1, either concrete system satisfies a given $\overline{m}$-indexed formula iff the system satisfies the trivial instantiation of the formula. The atomic propositions and the epistemic modalities appearing in the trivial instantiation refer only to agents in $\{(i,j) \colon 1 \leq i \leq k, 1 \leq j \leq \overline{m}.i\}$. Therefore, the notion of stuttering simulation can be relaxed to refer only to these agents as follows.

Definition 3.15 ($\overline{m}$-stuttering simulation)**.** *An* $\overline{m}$-stuttering simulation *is a relation* $\sim_{\overline{m}ss} \subseteq G(\overline{n}) \times G(\overline{n}')$ *between* $\mathcal{S}(\overline{n})$ *and* $\mathcal{S}(\overline{n}')$ *if* $\iota(\overline{n}) \sim_{\overline{m}ss} \iota(\overline{n}')$ *and whenever* $g \sim_{\overline{m}ss} g'$ *the following conditions hold:*

(i) $V(\overline{n})(g) \cap X = V(\overline{n}')(g') \cap X$*, where* $X = \bigcup_{1\leq i\leq k} \{(p,j) \colon p \in AP_i \wedge 1 \leq j \leq \overline{m}.i\}$*;*

(ii) If $\mathcal{K}_i^j(g, g^1)$*, for* $(i,j) \in \mathcal{A}(\overline{m})$*, then* $\mathcal{K}'^j_i(g', g'^1)$ *for some* g'^1 *such that* $g^1 \sim_{\overline{m}ss} g'^1$*.*

(iii) For every path $\pi \in \Pi(g)$*, there is a path* $\pi' \in \Pi(g')$*, a partition* $B_1, B_2, \ldots$ *of the states in* π*, and a partition* $B'_1, B'_2, \ldots$ *of the states in* π' *such that for each* $j \geq 1$*,* B_j *and* B'_j *are nonempty and finite, and every state in* B_j *is related by* $\sim_{\overline{m}ss}$ *to every state in* B'_j*.*

We say that a concrete system $\mathcal{S}(\overline{n}')$ $\overline{m}$-stuttering simulates a concrete system $\mathcal{S}(\overline{n})$, denoted $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{n}')$, if there is an $\overline{m}$-stuttering-simulation relation between $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{n}')$. Indexed ACTL*K$\backslash X$ formulae are preserved under $\overline{m}$-stuttering simulation.

Theorem 3.1. *Let $\forall_{\overline{v}}\phi(\overline{v})$ be an $\overline{m}$-indexed formula and $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{n}')$. Then, $\mathcal{S}(\overline{n}') \models \forall_{\overline{v}}\phi(\overline{v})$ implies $\mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v})$.*

Proof. By Lemma 3.1, it suffices to show that $\mathcal{S}(\overline{n}') \models \phi[trivial]$ implies $\mathcal{S}(\overline{n}) \models \phi[trivial]$. Suppose that $\mathcal{S}(\overline{n}') \models \phi[trivial]$. By Theorem 2.5, $\mathcal{S}(\overline{n}) \models \phi[trivial]$. $\square$

The result above enables us to access the truth of a specification on an arbitrarily big system by checking it on a smaller system. To do this, however, only conditions (i) and (iii) of Definition 3.15 have to be considered. Indeed, as the following shows, in the context of PIIS, the notion of stuttering simulation preserving temporal formulae [BCG88] is equivalent to the notion of stuttering simulation preserving temporal-epistemic formulae [LPQ10].

Proposition 3.1. *Let $\sim_x \subseteq G(\overline{n}) \times G(\overline{n}')$ be a relation between $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{n}')$ that satisfies conditions (i) and (iii) of $\overline{m}$-stuttering simulation. Then, $\sim_x$ is an $\overline{m}$-stuttering simulation between $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{n}')$.*

Proof. By assumption on $\sim_x$, we only have to show simulation requirement (ii). So, let $g \sim_x g'$ and $\mathcal{K}_i^j(g, g^1)$ for some agent $(i,j) \in \mathcal{A}(\overline{m})$. We show that there is a $g'^1 \in G(n')$ with $\mathcal{K}'^j_i(g', g'^1)$ and $g^1 \sim_x g'^1$. Consider a path $\pi \in \Pi(\iota(\overline{n}))$ such that $\pi(z) = g^1$ for some $z \geq 1$. As $\iota(\overline{n}) \sim_x \iota(\overline{n}')$, there is a path $\pi' \in \Pi(\iota(\overline{n}'))$ satisfying condition (iii) of $\overline{m}$-stuttering simulation. Thus, $\pi(z) \sim_x \pi'(z')$ for some $z' \geq 1$. Therefore, $ls_i^j(\pi(z)) = ls_i^j(\pi'(z'))$, since $\sim_x$ satisfies condition (i) of $\overline{m}$-stuttering simulation. Consequently, $\mathcal{K}'^j_i(g', \pi'(z'))$. So, $g'^1 = \pi'(z')$ is as required. Since (g, g') was arbitrary, we obtain $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{n}')$. $\square$

3.4.2 Equivalences preserving indexed ACTLK up to a level of depth

We begin with the definition of the temporal depth for an indexed CTLK formula by counting the temporal operators nested in the formula. We then define a simulation relation that preserves CTLK formulas up to a level of depth. In the following, we fix a PIS $\mathcal{S}$ of k roles, a concrete instantiation $\mathcal{S}(\overline{n}) = \Big(G(\overline{n}), \iota(\overline{n}), R(\overline{n}), (\mathcal{K}_i^j)_{(i,j)\in\mathcal{A}(\overline{n})}, V(\overline{n})\Big)$ of $\mathcal{S}$, and a concrete instantiation $\mathcal{S}(\overline{n}') = \Big(G(\overline{n}'), \iota(\overline{n}'), R(\overline{n}'), (\mathcal{K}'^j_i)_{(i,j)\in\mathcal{A}(\overline{n}')}, V(\overline{n}')\Big)$ of $\mathcal{S}$.

Definition 3.16 (Temporal depth). *The* temporal depth $td(\phi)$ *of an* $\overline{m}$*-indexed CTLK formula* ϕ *is inductively defined as follows.*

$$\begin{array}{llll}
td(\phi) & \triangleq & 0 & \text{if } \phi = (p, v); \\
td(\phi) & \triangleq & td(\psi) & \text{if } \phi = \neg\psi; \\
td(\phi) & \triangleq & \max(td(\psi_1), td(\psi_2)) & \text{if } \phi = \psi_1 \wedge \psi_2; \\
td(\phi) & \triangleq & td(\psi) & \text{if } \phi = K_i^v\psi; \\
td(\phi) & \triangleq & td(\psi) + 1 & \text{if } \phi = EX\psi; \\
td(\phi) & \triangleq & td(\psi) + 1 & \text{if } \phi = EG\psi; \\
td(\phi) & \triangleq & \max(td(\psi_1), td(\psi_2)) & \text{if } \phi = E(\psi_1 U \psi_2).
\end{array}$$

The notion of temporal depth finds a correspondence in the *cycle-stuttering simulation equivalence* that we now introduce. Intuitively, $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{n}')$ are cycle-stuttering simulation equivalent if they are behaviourally identical modulo stuttering of cyclic behaviours. To formally define this equivalence, we first introduce some preliminary definitions.

A substring $\delta = \pi(i), \dots, \pi(j)$ of a concrete path π in $\mathcal{S}(\overline{n})$ is said to be a cycle if $V(\overline{n})(\pi(i)) = V(\overline{n})(\pi(j))$. A cycle δ' in π corresponds to the x-th repetition of δ if $\delta' = \delta^x$, e.g., $\delta' = abababa = (aba)^3$. A cyclic-decomposition of π is an inductive partition of π into a sequence of *non-cyclic* and *cyclic* blocks. Each non-cyclic block is a finite sequence of states in π without repetitions. Each cyclic block corresponds to a (possibly infinite) repetition of a cycle in π and it can be further partitioned into cyclic and non-cyclic blocks. For example, consider the path $\pi = abcbcbdabcbcbda$. The partition $[[a][bcb]^2[da]]^2$ is a cyclic decomposition of π, where $[[a][bcb]^2[da]]^2$ is a cyclic block, $[a], [da]$ are non-cyclic blocks, and $[bcb]^2$ is a cyclic block. The partition $[a][bcb]^2[da][bcb]^2[da]$ is also a cyclic decomposition of π. Given a non-cyclic block C, we write $C(i)$ for the i-th state in C, and $|C|$ for the sequence's length. For a cyclic block $\mathfrak{C}^x$, we write $\mathfrak{C}^x[i]$ for the sequence of states corresponding to the i-th repetition of the cycle associated with $\mathfrak{C}^x$. If $\mathfrak{C}^x$ cannot be decomposed further, then we treat each $\mathfrak{C}^x[i]$ as a non-cyclic block, otherwise we treat it as a cyclic decomposition.

Definition 3.17. *We define the sequence of relations $\approx_0, \approx_1, \approx_2$ on pairs of states in $G(n) \times G(n')$ and the relations $\cong_0, \cong_1, \cong_2, \dots$ on pairs of paths in $\mathcal{S}(\overline{n}) \times \mathcal{S}(\overline{n}')$.*

We start by defining two states g and g' to be $\approx_0$-related whenever the following hold:

- $V(\overline{n})(g) = V(\overline{n}')(g')$;
- if $\mathcal{K}_i^j(g, g^1)$, for $(i, j) \in \mathcal{A}(\overline{m})$, then $\mathcal{K'}_i^j(g', g'^1)$ for some g'^1 with $g^1 \approx_0 g'^1$.

Then, for all $d \geq 0$, we define $\cong_d$ as follows. Two paths π and π' (either both finite or both infinite) are $\cong_d$-related if there is a cyclic decomposition $B_1B_2B_3\dots$ of π and a cyclic decomposition $B'_1B'_2B'_3\dots$ of π' such that for all $i \geq 1$, we have that $B_i \propto_d B'_i$, where $B_i \propto_d B'_i$ if:

- B_i and B'_i are blocks of the same type;
- If B_i and B'_i are two non-cyclic blocks C_i and C'_i, respectively, then: (i) $|C_i| = |C'_i|$; (ii) for each $1 \leq j \leq |C_i|$, we have that $C_i(j) \approx_d C'_i(j)$;
- If B_i and B'_i are two cyclic blocks $\mathfrak{C}_i^{x_i}$ and $\mathfrak{C'}_i^{x'_i}$, respectively, then: (i) either $x_i = x'_i$ or $x_i > d, x'_i > d$; (ii) for every pair $\mathfrak{C}_i^{x_i}[z], \mathfrak{C'}_i^{x'_i}[z']$ of repetitions of $\mathfrak{C}_i^{x_i}$ and $\mathfrak{C'}_i^{x'_i}$, we have the following:
  - if $x_i - z = x'_i - z'$, then $\mathfrak{C}_i^{x_i}[z] \propto_d \mathfrak{C'}_i^{x'_i}[z']$;
  - if $x_i - z \neq x'_i - z'$, then $\mathfrak{C}_i^{x_i}[z] \propto_{d'} \mathfrak{C'}_i^{x'_i}[z']$, where $d' = \min(d, x_i - z, x'_i - z')$.

So, for two non-cyclic blocks C_i and C'_i, $C_i \propto_d C'_i$ if: (i) the blocks have the same length; and (ii) each pair $(C_i(j), C'_i(j))$ of states is $\approx_d$-related. For two cyclic blocks $\mathfrak{C}_i^{x_i}$ and $\mathfrak{C'}_i^{x'_i}$, $\mathfrak{C}_i^{x_i} \propto_d \mathfrak{C'}_i^{x'_i}$ if: (i) the blocks agree up to d on the number of repetitions of their associated cycles; and (ii) each pair of repetitions $(\mathfrak{C}_i^{x_i}[z], \mathfrak{C'}_i^{x'_i}[z'])$ is $\propto_d$-related if they agree on the number of repetitions they can "see" ahead in their blocks; otherwise it is $\propto_{d'}$-related, where d' is the greatest number of repetitions in $\{1, \dots, d\}$ such that $\mathfrak{C}_i^{x_i}[z]$ and $\mathfrak{C'}_i^{x'_i}[z']$ can both "see" d' repetitions ahead in their blocks. Figure 3.9 exemplifies the relations between two cyclic blocks with 5 and 3 repetitions, respectively, and for $d = 2$.

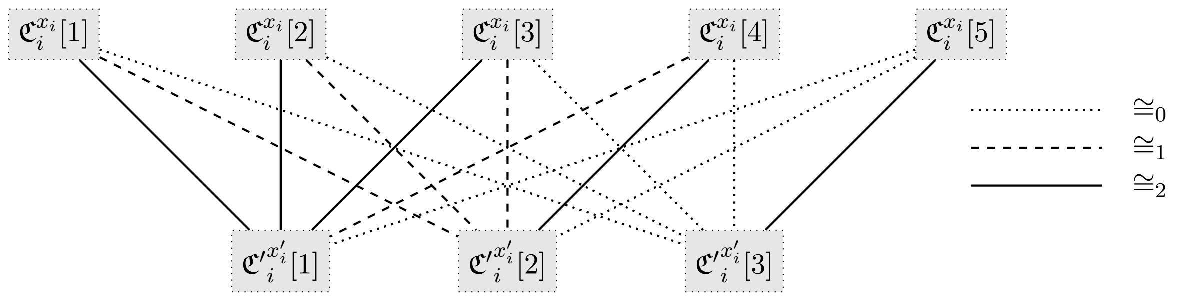

$\mathfrak{C}_i^{x_i}[1]$
$\mathfrak{C}_i^{x_i}[2]$
$\mathfrak{C}_i^{x_i}[3]$
$\mathfrak{C}_i^{x_i}[4]$
$\mathfrak{C}_i^{x_i}[5]$
$\mathfrak{C}'^{x'_i}_i[1]$
$\mathfrak{C}'^{x'_i}_i[2]$
$\mathfrak{C}'^{x'_i}_i[3]$
$\cong_0$
$\cong_1$
$\cong_2$

Figure 3.9: Cycle-stuttering relations between cyclic blocks for a temporal depth of 2.

Finally, we define $\approx_{d+1}$ as follows. Two states g and g' are $\approx_{d+1}$-related if for every path $\pi \in \Pi(g)$ there is a path $\pi' \in \Pi(g')$ such that $\pi \cong_d \pi'$;

Definition 3.18 (Cycle-stuttering simulation). *A concrete system $\mathcal{S}(\overline{n}')$ cycle stuttering simulates up to degree d a concrete system $\mathcal{S}(\overline{n}')$, denoted by $\mathcal{S}(\overline{n}) \leq_{d.css} \mathcal{S}(\overline{n}')$, if $\iota \approx_d \iota'$ for every pair (ι, ι') of initial states in $I(\overline{n}) \times I(\overline{n}')$.*

Indexed ACTLK formulas with temporal nesting up to d are preserved by cycle-stuttering simulation of degree $d+1$. To prove this, we first show the following.

Proposition 3.2. *Let $\pi \cong_d \pi'$. Then, the following hold:*

1. *for each state $\pi(i)$, there is a state $\pi'(i')$ such that $\pi(i) \approx_d \pi'(i')$;*
2. *if $\pi(i) \approx_d \pi'(i')$, then for each state $\pi(j)$ in $\pi(1), \ldots, \pi(i)$, there is a state $\pi'(j')$ in $\pi'(1), \ldots, \pi'(i')$ such that $\pi(j) \approx_d \pi'(j')$.*

Proof. Let $C_1, \mathfrak{C}_1^{x_1}, \ldots, C'_1, \mathfrak{C}'^{x'_1}_1, \ldots$ be the associated cyclic decompositions of π and π', respectively. By definition of $\cong_d$, it suffices to show the following:

1. each repetition $\mathfrak{C}_i^{x_i}[z]$ of a cyclic block $\mathfrak{C}_i^{x_i}$ is $\propto_d$-related to a repetition $\mathfrak{C}'^{x'_i}_i[z']$ of the cyclic block $\mathfrak{C}'^{x'_i}_i$;
2. if $\mathfrak{C}_i^{x_i}[z] \propto_d \mathfrak{C}'^{x'_i}_i[z']$, then for each j with $1 \leq j \leq z$, there is a j' with $1 \leq j' \leq z'$ and $\mathfrak{C}_i^{x_i}[j] \propto_d \mathfrak{C}'^{x'_i}_i[j']$.

Assume, without loss of generality, that $x_i \leq x'_i$. Then, each repetition $\mathfrak{C}_i^{x_i}[1], \ldots, \mathfrak{C}_i^{x_i}[x_i]$ of $\mathfrak{C}_i^{x_i}$ is $\propto_d$-related to the repetition $\mathfrak{C}'^{x'_i}_i[x'_i - x_i + 1], \ldots, \mathfrak{C}'^{x'_i}_i[x'_i]$, respectively, of $\mathfrak{C}'^{x'_i}_i$. And each repetition $\mathfrak{C}'^{x'_i}_i[1], \ldots, \mathfrak{C}'^{x'_i}_i[x'_i - x_i]$ of $\mathfrak{C}'^{x'_i}_i$ is $\propto_d$-related to the repetition $\mathfrak{C}^{x_i}_i[1]$ of $\mathfrak{C}^{x_i}_i$ (as illustrated in Figure 3.9). □

Theorem 3.2. *Let $\forall_{\overline{v}}\phi(\overline{v})$ be an $\overline{m}$-indexed ACTLK formula with $td(\forall_{\overline{v}}\phi(\overline{v})) \leq d$. Let $\mathcal{S}(\overline{n}) \leq_{(d+1).css} \mathcal{S}(\overline{n}')$. Then, $\mathcal{S}(\overline{n}') \models \forall_{\overline{v}}\phi(\overline{v})$ implies $\mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v})$.*

Proof. Let $(g, g') \in G(\overline{n}) \times G(\overline{n}')$. We show that if $g \approx_{d+1} g'$, then $g' \models \forall_{\overline{v}}\phi(\overline{v})$ implies $g \models \forall_{\overline{v}}\phi(\overline{v})$, for any $\overline{m}$-indexed ACTLK formula $\forall_{\overline{v}}\phi(\overline{v})$ with $td(\forall_{\overline{v}}\phi(\overline{v})) \leq d$. To do this, by Lemma 3.1, it suffices to prove that if $g \approx_{d+1} g'$, then $g' \models \phi[trivial]$ implies $g \models \phi[trivial]$.

The proof is by induction on d in which each case is shown by structural induction on $\phi[trivial]$.

For the base step, we have $d = 0$ and $g \approx_1 g'$. Then, $g \approx_0 g'$. The case of $\phi[trivial] \in AP_i \times \{1, \ldots, \overline{m}.i\}$ follows directly from the definition of $\approx_0$. The cases of $\neg$ and $\wedge$ are straightforward.

Consider $\phi[trivial] = K^j_i\psi$, and assume that $g' \models \phi[trivial]$. So, $g'^1 \models \psi$ for every g'^1 with $\mathcal{K}'^j_i(g', g'^1)$. Assume $g^1 \in G(\overline{n})$ with $\mathcal{K}^j_i(g, g^1)$. We have to show that $g^1 \models \psi$. By definition of $\approx_0$, there is a $g'^1 \in G(\overline{n}')$ with $\mathcal{K}'^j_i(g', g'^1)$ and $g^1 \approx_0 g'^1$. Since $g'^1 \models \psi$, the inductive hypothesis gives $g^1 \models \psi$. As g^1 was arbitrary, it follows that $g \models K^j_i\psi$.

For the inductive step, assume that whenever $g \approx_d g'$, $g' \models \phi[trivial]$ implies $g \models \phi[trivial]$, for any $\overline{m}$-indexed ACTLK formula $\phi(\overline{v})$ with $td(\phi(\overline{v})) \leq d-1$. Suppose that $g \approx_{d+1} g'$. We have to show that $g' \models \phi[trivial]$ implies $g \models \phi[trivial]$, for any $\overline{m}$-indexed ACTLK formula $\phi(\overline{v})$ with $td(\phi(\overline{v})) \leq d$. For $\phi[trivial] \in AP_i \times \{1, \ldots, \overline{m}.i\}$, and for the cases of $\neg, \wedge$, the thesis follows along the lines of the main base step.

Consider $\phi[trivial] = AX\psi$. Then, there is a path $\pi' \in \Pi(g')$ with $\pi'(2) \models \psi$. By the definition of $\approx_{d+1}$, there is a path $\pi \in \Pi(g)$ such that $\pi \cong_d \pi'$. Hence, $\pi(2) \approx_d \pi'(2)$. So, by the inductive hypothesis, $\pi(2) \models \psi$. Thus, $g \models EX\psi$.

Assume $\phi[trivial] = E(\psi_1 U \psi_2)$ and $g' \models E(\psi_1 U \psi_2)$. Then, there is a path $\pi' \in \Pi(g')$ and an $i' \geq 1$ such that $\pi'(i') \models \psi_2$ and $\pi'(j') \models \psi_1$ for all $1 \leq j' < i'$. By the definition of $\approx_{d+1}$, there is a path $\pi \in \Pi(g)$ with $\pi \cong_d \pi'$. By Proposition 3.2, there is a state $\pi(i)$ in π with $\pi(i) \approx_d \pi'(i')$; further, for every state $\pi(j)$ with $1 \leq j < i$, there is a state $\pi'(j')$ with

$1 \leq j' < i'$ and $\pi(j) \approx_d \pi'(j')$. Thus, by the inductive hypothesis, $\pi(i) \models \psi_2$, and $\pi(j) \models \psi_1$ for all $1 \leq j < i$. So, $g \models E(\psi_1 U \psi_2)$.

Finally, for the case of $\phi[trivial] = K_i^j \psi_1$, assume that $g' \models K_i^j \psi_1$. Thus, $g'^1 \models \psi$ for every g'^1 with $\mathcal{K}'^j_i(g', g'^1)$. Let $g^1 \in G(\overline{n})$ with $\mathcal{K}_i^j(g, g^1)$. We have to show that $g^1 \models \psi$. As every pair of initial states in $G(\overline{n}) \times G(\overline{n}')$ are $\approx_{d+1}$-related, every pair of paths originating from the initial states of the two models are $\cong_d$-related. Therefore, by Proposition 3.2, there is a state $g'^2 \in G(\overline{n}')$ with $g^1 \approx_d g'^2$. Hence, $g^1 \approx_0 g'^2$, and hence $ls_i^j(g^1) = ls_i^j(g'^2)$, as the states satisfy the same atomic propositions. Therefore, $\mathcal{K}'^j_i(g', g'^2)$, and therefore $g'^2 \models \psi$. By the inductive hypothesis, $g^1 \models \psi$. As g^1 was arbitrary, it follows that $g \models K_i^j \psi$. □

We call two concrete systems $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{n}')$ cycle-stuttering simulation equivalent up to degree d if $\mathcal{S}(\overline{n}) \leq_{(d+1).css} \mathcal{S}(\overline{n}')$ and $\mathcal{S}(\overline{n}') \leq_{(d+1).css} \mathcal{S}(\overline{n})$. The following theorem relates indexed ACTLK with cycle-stuttering simulation equivalence.

Theorem 3.3. *Let $\phi(\overline{v})$ be an $\overline{m}$-indexed ACTLK formula with $td(\phi(\overline{v})) \leq d$. Let $\mathcal{S}(\overline{n}) \leq_{(d+1).css} \mathcal{S}(\overline{n}')$ and $\mathcal{S}(\overline{n}') \leq_{(d+1).css} \mathcal{S}(\overline{n})$. Then, $\mathcal{S}(\overline{n}) \models \phi(\overline{v})$ iff $\mathcal{S}(\overline{n}') \models \phi(\overline{v})$.*

Proof. The theorem follows from Theorem 3.2. □

3.5 Summary

This chapter introduced parameterised interpreted systems and parameterised interleaved interpreted systems.

A parameterised interpreted system *PIS* of k roles gives a concise description of an unbounded collection $\{PIS(\overline{n}) : \overline{n} \in \mathbb{N}^k\}$ of concrete interpreted systems. Each system is built from $\overline{n}.i$ identical agents, for each role i, and from the concrete environment corresponding to the $\overline{n}$'th instantiation of the environment template. Differently from the standard treatment of interpreted systems, the evolution function of a concrete agent does not depend on which agent performed which action (i.e., the joint action), but it depends on the set of actions performed by all the agents.

Similarly, a parameterised interleaved interpreted system $PIIS$ of k roles gives a concise description of an unbounded collection $\{PIIS(\overline{n})\colon \overline{n} \in \mathbb{N}^k\}$ of concrete interleaved interpreted systems. Each system is built from $\overline{n}.i$ identical agents, for each role i, and from the concrete environment corresponding to the $\overline{n}$'th instantiation of the environment template. The concrete agents may evolve asynchronously, communicate with the environment via agent-environment actions, synchronise with the agents of the same role via role-synchronous actions, synchronise with all the agents in the system via global-synchronous actions, and communicate with an agent performing another role via multi-role actions.

Further, this chapter introduced the specification language indexed CTL*K. The logic extends CTL*K by introducing indexed atomic propositions and indexed epistemic modalities. This allows for properties to be expressed independently of the number of agents in a UMAS.

Finally, the notions of $\overline{m}$-stuttering simulation and cycle-stuttering simulation were defined on PIIS and PIS, respectively. The former simulation was shown to preserve indexed ACTL*K$\backslash X$ formulae. The latter simulation was shown to preserve indexed ACTLK formulae up to a level of temporal depth.

Chapter 4

Verifying parameterised interpreted systems

This chapter develops a parameterised model checking procedure for the verification of unbounded multiagent systems represented in the formalism of parameterised interpreted systems. Section 4.1 gives a formal definition of the parameterised model checking problem and of the notion of cutoffs for parameterised interpreted systems. Section 4.2 introduces the `PIS` procedure. The procedure identifies a cutoff for a given system and a given specification. A cutoff expresses the number of agents that is sufficient to consider when evaluating a given specification. Following the cutoff identification, the procedure solves the parameterised model checking problem by checking all concrete systems up to the cutoff. Finally, Section 4.3 applies the procedure to the autonomous robot example and the Beta swarm aggregation algorithm.

4.1 Parameterised model checking problem

We outline parameterised interpreted systems as presented in Section 3.1. We then define the parameterised model checking problem for PIS. Finally, we define the notion of cutoffs in the context of PIS.

A PIS is a tuple

$$PIS = (\mathcal{T}, \mathcal{E}, \mathcal{I}, \mathcal{V})$$

where $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\} = \{(L_1, Act_1, P_1, t_1), \ldots, (L_k, Act_k, P_k, t_k)\}$ is a set of $k \geq 1$ agent templates and $\mathcal{E} = (L_\mathcal{E}, Act_\mathcal{E}, P_\mathcal{E}, t_\mathcal{E})$ is an environment template. An agent template is associated with a set L_i of template states, a set Act_i of template actions, a template protocol $P_i : L_i \to \mathcal{P}(Act_i)$, and a template transition function $t_i : L_i \times Act_i \times \mathcal{P}(\bigcup_{\mathcal{T}_i} Act_i) \times Act_\mathcal{E} \to L_i$. The environment template is similarly associated with a set $L_\mathcal{E}$ of template states, a set $Act_\mathcal{E}$ of template actions, a template protocol $P_\mathcal{E} : L_\mathcal{E} \to \mathcal{P}(Act_\mathcal{E})$, and a template transition function $t_\mathcal{E} : L_\mathcal{E} \times Act_\mathcal{E} \times \mathcal{P}(\bigcup_{\mathcal{T}_i} Act_i) \to L_\mathcal{E}$. The definition of a PIS concludes with the description of a set $\mathcal{I} = I_1 \times \ldots I_k \times I_\mathcal{E}$ of initial states for the templates, and a set of valuation functions on the template states $\mathcal{V} = \{V_i : L_i \to \mathcal{P}(AP_i) \colon 1 \leq i \leq k\}$.

Given a value $\overline{n} = (n_1, \ldots, n_k)$ of the system's parameter, the concrete system $PIS(\overline{n})$ is the interpreted system

$$PIS(\overline{n}) = \left((L_i^j, Act_i^j, P_i^j, t_i^j)_{(i,j)\mathcal{A}(\overline{n})}, (L_E, Act_E, P_E, t_E), I, V\right).$$

$PIS(\overline{n})$ results from the composition of $\overline{n}.i$ instantiations $\{(i,1), \ldots, (i, \overline{n}.i)\}$ of each agent template $\mathcal{T}_i$ and an instantiation of the environment template. Given $\overline{x} \leq \overline{n}$, we write $\mathcal{A}(\overline{x})$ for the set $\mathcal{A}(\overline{x}) = \{(i,j) \colon 1 \leq i \leq k, 1 \leq j \leq \overline{x}.i\}$ of concrete agents. For each concrete system $PIS(\overline{n})$ we can associate a temporal-epistemic model $\mathcal{S}_{PIS(\overline{n})} = \left(G(\overline{n}), I(\overline{n}), R(\overline{n}), (\mathcal{K}_i^j)_{(i,j)\in\mathcal{A}(\overline{n})}, V(\overline{n})\right)$ as standard (see Section 2.1.1). When $PIS(\overline{n})$ is clear from the context we simply write $\mathcal{S}(\overline{n})$ for $\mathcal{S}_{PIS(\overline{n})}$. We assume that the transition relation for each concrete system is serial [1].

We now proceed to state the parameterised model checking problem for PIS.

Definition 4.1 (Parameterised model checking problem for PIS). *Given a PIS $\mathcal{S}$ and an $\overline{m}$-indexed ACTLK formula $\forall_{\overline{v}}\phi(\overline{v})$, the parameterised model checking problem concerns establishing whether or not the following holds:*

$$\forall \overline{n} \geq \overline{m}. \mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v})$$

[1]A serial transition relation can be enforced by the templates' descriptions by assuming a *null* action that is enabled at each local state for each of the templates. Upon performing the *null* action, an agent remains in its current state irrespective of the other agents' actions.

If the above holds, then $\forall_{\overline{v}}\phi(\overline{v})$ is said to be satisfied by $\mathcal{S}$. This is denoted by $\mathcal{S} \models \forall_{\overline{v}}\phi(\overline{v})$.

In other words, differently from the standard model checking problem, the parameterised model checking problem involves establishing whether a specification is satisfied on an unbounded number of systems resulting from the instantiations of the agent templates. This is a task that in principle involves an unbounded state space which is intractable for traditional model checking techniques. Cutoffs have been studied in the context of reactive systems to circumvent this difficulty by reducing the number of systems to consider [EK00, EN95, HBR09, KKW10]. A cutoff for a system is the number of components that is sufficient to consider when evaluating a given specification.

Definition 4.2 (MAS cutoff). *Given a PIS $\mathcal{S}$ composed of k roles and an $\overline{m}$-indexed ACTLK formula $\forall_{\overline{v}}\phi(\overline{v})$, a k-tuple $\overline{c} \in \mathbb{N}^k$ is said to be a* MAS cutoff *if the following holds:*

$$\forall \overline{m} \leq \overline{x} \leq \overline{c}.\, \mathcal{S}(\overline{x}) \models \forall_{\overline{v}}\phi(\overline{v}) \text{ if and only if } \forall \overline{n} \geq \overline{m}.\, \mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v})$$

We say that $\mathcal{S}$ admits a cutoff for $\forall_{\overline{v}}\phi(\overline{v})$ and we call $\mathcal{S}(\overline{c})$ the cutoff system*.*

By definition, given a cutoff for a system, the parameterised model checking problem can be reduced to model checking all concrete systems up to the cutoff system. The aim of this chapter is to develop a sound and complete procedure that takes as input a PIS $\mathcal{S}$ and an $\overline{m}$-indexed formula $\forall_{\overline{v}}\phi(\overline{v})$, and returns a cutoff for $\mathcal{S}$ and $\forall_{\overline{v}}\phi(\overline{v})$.

4.2 The `PIS` procedure

This section introduces the `PIS` procedure. The procedure tackles the verification of parameterised interpreted systems against indexed ACTLK formulae.

4.2.1 Overview

Given a PIS $\mathcal{S}$ and an $\overline{m}$-indexed ACTLK formula $\forall_{\overline{v}}\phi(\overline{v})$, the procedure identifies a cutoff $\overline{c}$ for $\mathcal{S}$ and $\forall_{\overline{v}}\phi(\overline{v})$. Following this, the procedure checks every concrete system up to $\mathcal{S}(\overline{c})$ against

$\forall_{\overline{v}}\phi(\overline{v})$. If the formula is satisfied on each of the systems checked, then the procedure concludes that the formula is satisfied on every concrete system. The cutoff identification is performed by means of the following three steps:

1. In the first step, an abstract model is built. The model can simulate any concrete system from $\mathcal{S}$ of arbitrary size. Therefore, the satisfaction of $\forall_{\overline{v}}\phi(\overline{v})$ on the abstract model entails the satisfaction of $\forall_{\overline{v}}\phi(\overline{v})$ on every concrete system. However, if $\forall_{\overline{v}}\phi(\overline{v})$ is not satisfied by the abstract model, then it does not necessarily follow that there is a concrete system falsifying the formula. This is because the abstract model may contain spurious paths, i.e., paths that do not correspond to any concrete behaviour. Spurious paths correspond to cycles that are repeated infinitely often. Although a concrete system can perform these cycles, it can do so only for a finite number of repetitions, the exact number depending on the size of the system.

2. Following step 1, in step 2, a pruned computation tree of the abstract model is built. The tree contains all and only the abstract paths that do not contain a cycle repeated for more than $td(\forall_{\overline{v}}\phi(\overline{v})) + 1$ times, where $td(\forall_{\overline{v}}\phi(\overline{v}))$ is the temporal depth of $\forall_{\overline{v}}\phi(\overline{v})$. Intuitively, for any $d > td(\forall_{\overline{v}}\phi(\overline{v})), d' > td(\forall_{\overline{v}}\phi(\overline{v}))$, $\forall_{\overline{v}}\phi(\overline{v})$ cannot distinguish between d and d' repetitions of a given cycle (see Section 3.4 for a detailed discussion).

3. In step 3, the number of agents that enable a concrete system to simulate the tree constructed in step 2 is computed. Intuitively, any concrete system that can simulate the tree admits all possible behaviour that may alter the satisfaction status of $\forall_{\overline{v}}\phi(\overline{v})$. The number computed in this step is the cutoff.

In the following sections we give a detailed description of the above steps.

4.2.2 Step 1: abstraction

Given a PIS $\mathcal{S}$ and an $\overline{m}$-indexed formula $\forall_{\overline{v}}\phi(\overline{v})$, the abstract model is generated by counter abstraction principles with a threshold equal to 1. An abstract state is built from a concrete component and an abstract component for each agent template. The concrete component consists of the local states of the concrete agents in $\mathcal{A}(\overline{m})$. The abstract component of each agent template $\mathcal{T}_i$ consists of a set of template states representing the projection of the tuple of local states of all the agents performing role i other than the ones in $\mathcal{A}(\overline{m})$ into a set. Note that the agents $\mathcal{A}(\overline{m})$ are not abstracted. This is because the atomic propositions and the epistemic modalities appearing in the trivial instantiation of $\forall_{\overline{v}}\phi(\overline{v})$ refer to all and only these agents. Transitions from an abstract state represent transitions enabled from any concrete state that is represented by said abstract state. Each abstract transition is labelled by the template transitions taking place.

Example 4.1. *Figure 4.1 exemplifies these ideas on a 2-indexed formula, and on a global transition of a concrete system of 1 template and with ten participants. The figure uses different styles of rectangles to indicate different template local states. The concrete global state shown at the top left corner consists of three agents in the local state represented by the gray rectangle, two agents in the local state represented by the white rectangle with solid borders, four agents in the local state represented by the white rectangle with dotted borders, and one agent in the local state represented by the black rectangle. The state is counter abstracted to the abstract state shown in the bottom left corner. The concrete component for the state is the tuple of local states for the first two agents. The abstract component for the state is the set of local states for the agents 3,...,10; i.e., the set containing the template local state represented by the gray rectangle, the template local state represented by the white rectangle with solid borders, the template local state represented by the white rectangle with dotted borders, and the template local state represented by the black rectangle. The concrete global state in the top right corner, which is reachable in one step from the state in the top left corner, is similarly abstracted to the abstract state in the bottom right corner. Finally, note the abstract transition representing the concrete one is labelled by the template transitions taking place in the concrete global transition; i.e., there is an agent in* $\{3, \dots, 10\}$*, namely 3, in the local state represented by the gray rectangle that moves to the local state represented by the white rectangle with solid borders, and there is an agent, namely 4, in the local state represented by the white rectangle with solid borders that remains in the same state, and so forth.*

We now formally define the abstract model.

Definition 4.3 (Abstract model)**.** *Given a PIS $\mathcal{S}$ of k roles and $\overline{m} \in \mathbb{N}^k$, the abstract model $\hat{\mathcal{S}}(\overline{m})$ is defined as a tuple $\hat{\mathcal{S}}(\overline{m}) = \left(\hat{G}(\overline{m}), \hat{I}(\overline{m}), \hat{R}(\overline{m})\right)$, where*

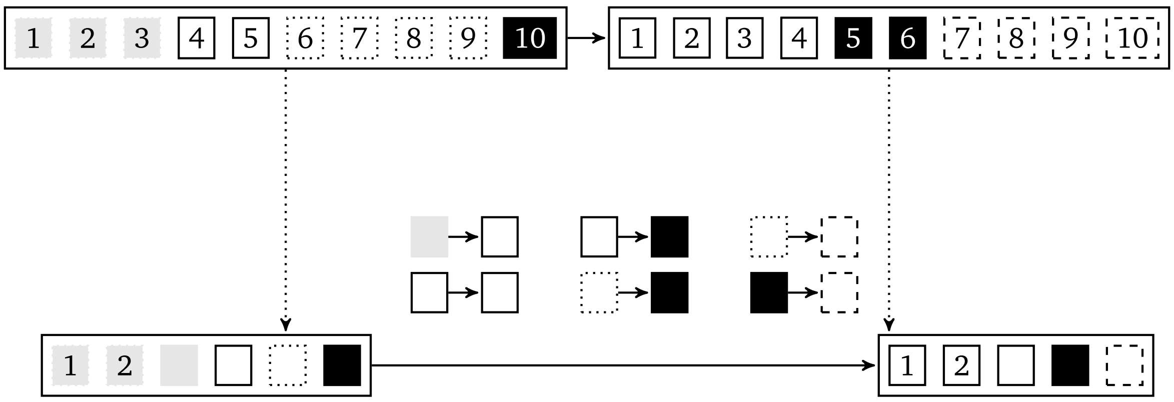


Figure 4.1: Abstraction realised by the `PIS` procedure.

- *$\hat{G}(\overline{m}) \subseteq \left(L_1^1 \times \ldots \times L_k^{\overline{m}.k} \times L_E\right) \times ((\mathcal{P}(L_1) \setminus \{\emptyset\}) \times \ldots \times (\mathcal{P}(L_k) \setminus \{\emptyset\}))$ is the set of abstract states.*

- *$\hat{I}(\overline{m})$ is the set of initial abstract states. We have that $((l_1^1, \ldots, l_k^{\overline{m}.k}, l_E), (X_1, \ldots, X_k)) \in \hat{I}(\overline{m})$ iff*

  - $l_1^1 \in I_1 \times \{1\}, \ldots, l_k^{\overline{m}.k} \in I_k \times \{\overline{m}.k\}$.
  - $l_E \in I_E$.
  - $X_1 \subseteq I_1 \setminus \{\emptyset\}, \ldots, X_k \subseteq I_k \setminus \{\emptyset\}$.

- *$\hat{R}(\overline{m}) \subseteq \hat{G}(\overline{m}) \times \Lambda_1 \times \ldots \times \Lambda_k \times \hat{G}(\overline{m})$ is the abstract transition relation for the sets of labels*

$$\Lambda_1 = \mathcal{P}(L_1 \times Act_1 \times \mathcal{P}(All\_Act) \times L_1) \setminus \{\emptyset\}, \ldots,$$
$$\Lambda_k = \mathcal{P}(L_k \times Act_k \times \mathcal{P}(All\_Act) \times L_k) \setminus \{\emptyset\},$$

  where $All\_Act = \bigcup_{1 \le i \le k} Act_i$ is the set of all actions for the agent templates.

An abstract state $\gamma = ((l_1^1, \ldots, l_k^{\overline{m}.k}, l_E), (X_1, \ldots, X_k))$ consists of a concrete component $\gamma.c = (l_1^1, \ldots, l_k^{\overline{m}.k}, l_E)$ and an abstract component $\gamma.\hat{a} = (X_1, \ldots, X_k)$. γ represents any global state g in any concrete system $\mathcal{S}(\overline{n})$ (with $\overline{n} > \overline{m}$) in which:

- the environment is at local state l_E;
- the agents $(1,1), \ldots, (k, \overline{m}.k)$ are at local states $l_1^1, \ldots, l_k^{\overline{m}.k}$, respectively;

- $X_i = \left\{ tls_i^j(g) \colon j \in \{\overline{m}.i+1, \ldots, \overline{n}.i\} \right\}$ is the projection of the tuple of template local states for the agents $(i, \overline{m}.i+1), \ldots, (i, \overline{n}.i)$ into a set.

Thus $\gamma.c$ encodes the atomic propositions on which an $\overline{m}$-indexed formula is built, and $\gamma.\hat{a}$ encodes how an arbitrary number of agents may interfere with the state of $\gamma.c$. To see this, consider a transition $(\gamma, \Xi_1, \ldots, \Xi_k, \gamma') \in \hat{R}(\overline{m})$, representing a set of transitions between the concrete states represented by γ and γ'. Each set of labels Ξ_i indicates the template transitions enabling the agents performing role i to participate in a global transition. In other words, a tuple $(l, a, A, l') \in \Xi_i$ indicates that in a global transition at least one agent of role i is at state l; this agent performs action a, all agents in the system jointly perform the actions in the set A of actions, and the agent moves to state l'.

Given an abstract state γ, we write $ls_i^j(\gamma.c)$ for the local state of agent (i,j) in γ. By $tls_i^j(\gamma.c)$, we express the template local state of agent (i,j) in γ. We use $\gamma.\hat{a}.i$ to denote the abstract component of agent template i in $\gamma.\hat{a}$. For a set $\Xi_i \subseteq \Lambda_i$, let $Act(\Xi_i) = \{a \colon \exists l, l', A.\, (l, a, A, l') \in \Xi_i\}$ be the set of actions performed by Ξ_i. The construction of the abstract model concludes with the definition of the abstract transition relation.

Definition 4.4 (Abstract transition relation). *The construction of $\hat{S}(\overline{m})$ is completed by defining $\hat{R}(\overline{m})$ as follows: $(\gamma, \Xi_1, \ldots, \Xi_k, \gamma') \in \hat{R}(\overline{m})$ iff there is a joint action $\overline{a} \in Act_1^1 \times \ldots \times Act_k^{\overline{m}.k} \times Act_E$ such that:*

1. *A cocnrete transition represented by $(\gamma, \Xi_1, \ldots, \Xi_k, \gamma')$ is valid for the environment:*

    - *$la_E(\overline{a}) \in P_{\mathcal{E}}(ls_E(\gamma.c))$;*
    - *$t_{\mathcal{E}}(ls_E(\gamma.c), la_E(\overline{a}), A) = ls_E(\gamma'.c)$, where*

$$A = Act(\Xi_1) \cup \ldots \cup Act(\Xi_k) \cup \left\{ la_i^j(\overline{a}) \colon (i,j) \in \mathcal{A}(\overline{m}) \right\}.$$

2. *A concrete transition represented by $(\gamma, \Xi_1, \ldots, \Xi_k, \gamma')$ is valid for each concrete agent: for all $(i,j) \in \mathcal{A}(\overline{m})$, we have that:*

    - *$la_i^j(\overline{a}) \in P_i(tls_i^j(\gamma.c))$;*

- $t_i(tls_i^j(\gamma.c), la_i^j(\overline{a}), A, la_E(\overline{a})) = tls_i^j(\gamma'.c)$, *where*

$$A = Act(\Xi_1) \cup \ldots \cup Act(\Xi_k) \cup \left\{ la_i^j(\overline{a}) \colon (i,j) \in \mathcal{A}(\overline{m}) \right\}.$$

3. *A concrete transition represented by* $(\gamma, \Xi_1, \ldots, \Xi_k, \gamma')$ *is valid for the agents represented by each abstract component* $\gamma.\hat{a}.i$*: for all* $(l, a, A, l') \in \Xi_i$*, we have that:*

    - $l \in \gamma.\hat{a}$;
    - $l' \in \gamma'.\hat{a}$;
    - $a \in P_i(l)$;
    - $t_i(l, a, A, la_E(\overline{a})) = l'$, *where*

$$A = Act(\Xi_1) \cup \ldots \cup Act(\Xi_k) \cup \left\{ la_i^j(\overline{a}) \colon (i,j) \in \mathcal{A}(\overline{m}) \right\}.$$

4. *For every* $1 \leq i \leq k$ *and every* $l \in \gamma.\hat{a}.i$*, there is a transition label in* Ξ_i *to a state* $l' \in \gamma'.\hat{a}.i$*, and for every* $l' \in \gamma'.\hat{a}.i$*, there is a transition label in* Ξ_i *from a state* $l \in \gamma.\hat{a}.i$*.*

A path in $\hat{\mathcal{S}}(\overline{m})$ is a sequence $\gamma^1(\Xi_1^1, \ldots, \Xi_k^1)\gamma^2, (\Xi_1^2, \ldots, \Xi_k^2), \gamma^3, \ldots$ with $(\gamma^i, \Xi_1^i, \ldots, \Xi_k^i, \gamma^{i+1}) \in \hat{R}(\overline{m})$, for every $i \geq 1$.

Example 4.2. *In Section 3.1.1 we represented the autonomous robots scenario in the formalism of parameterised interpreted systems. We encoded a robot by means of an agent template defined on local states of the form* (p, s, h)*, where* p *represents the position of the robot,* s *represents its sensor reading, and* h *represents whether it has halted (*$h = \top$*) or not (*$h = \bot$*). The agent template may perform one of the actions* $s_=, s_+, s_-$ *at each time step. The actions determine the value of the sensor at the next time step. The sensor will have the correct value whenever* $s_=$ *is performed; one value higher than the correct value whenever* s_+ *is performed; one value lower than the correct value whenever* s_- *is performed. Additionally, the agent template admits the* $halt$ *action. The action is performed whenever the value of its sensor is equal to or greater than 3.*

A fragment of the concrete system with two robots was given in Figure 3.3. Figure 4.2 gives a fragment of the abstract model for 1*-indexed formulae. A state of the abstract model is a triple denoting the concrete local state of agent 1, the set of template local states for all the other agents,*

$$\begin{tikzcd}
 & & \boxed{\mathbf{(00\bot)^1, \{(00\bot)\}, 0}} \arrow[dl, "{n_-, move, \Xi_1}"'] \arrow[dr, "{n_=, move, \Xi_2}"] & & \\
 & \boxed{\mathbf{(10\bot)^1, \{(11\bot), (12\bot)\}, 1}} \arrow[dl, "{n_=, move, \Xi_3}"'] \arrow[dr, "{n_+, move, \Xi_4}"'] & & \boxed{\mathbf{(11\bot)^1, \{(10\bot), (11\bot), (12\bot)\}, 1}} \arrow[dl, "{n_+, move, \Xi_5}"'] \arrow[dr, "{n_+, move, \Xi_6}"] & \\
\boxed{\mathbf{(22\bot)^1, \{(21\bot), (22\bot)\}, 2}} \arrow[d, "{n_+, move, \Xi_7}"'] & & \boxed{\mathbf{(23\bot)^1, \{(23\bot)\}, 2}} \arrow[d, "{halt, move, \Xi_8}"'] & & \boxed{\mathbf{(23\bot)^1, \{(22\bot), (23\bot)\}, 2}} \arrow[d, "{n_=, move, \Xi_9}"] \\
\boxed{\mathbf{(34\bot)^1, \{(32\bot), (33\bot), (34\bot)\}, 3}} & & \boxed{\mathbf{(23\top)^1, \{(23\top)\}, 3}} & & \boxed{\mathbf{(33\top)^1, \{(34\bot), (33\bot), (23\top)\}, 3}}
\end{tikzcd}$$

Figure 4.2: Fragment of the abstract model for the parameterised interpreted system of the autonomous robot scenario.

and the local state of the environment. Each transition in the figure is labelled with a triple indicating the action of agent 1, the action of the environment, and the set of labels of the abstract transition, respectively. Let $X \subseteq \{null_+, null_-, null_+, , null, halt\}$ denote an arbitrary subset of the set of actions admitted by the agent template. The sets of labels are as follows:

- $\Xi_1 = \{((00\bot), n_=, X, (11\bot)), ((00\bot), n_+, X, (12\bot))\}$;
- $\Xi_2 = \{((00\bot), n_-, X, (10\bot)), ((00\bot), n_=, X, (11\bot)), ((00\bot), n_+, X, (12\bot))\}$;
- $\Xi_3 = \{((11\bot), n_-, X, (21\bot)), ((12\bot), n_=, X, (22\bot))\}$;
- $\Xi_4 = \{((11\bot), n_+, X, (23\bot)), ((12\bot), n_+, X, (23\bot))\}$;
- $\Xi_5 = \{((10\bot), n_+, X, (23\bot)), ((11\bot), n_+, X, (23\bot)), ((12\bot), n_+, X, (23\bot))\}$;
- $\Xi_6 = \{((10\bot), n_=, X, (22\bot)), ((11\bot), n_+, X, (23\bot)), ((12\bot), n_+, X, (23\bot))\}$;
- $\Xi_7 = \{((21, \bot), n_-, X, (32, \bot)), ((21\bot), n_=, X, (33\bot)), ((22\bot), n_+, X, (34\bot))\}$;
- $\Xi_8 = \{((23\bot), halt, X, (23\top))\}$;
- $\Xi_9 = \{((22, \bot), n_+, X, (34\bot)), ((22\bot), n_=, X, (33\bot)), ((23\bot), halt, X, (23\top))\}$.

For instance, the abstract state $\gamma = \big((00\bot)^1, \{(00\bot)\}, 0\big)$ represents any concrete state in which agent 1 is in local state $(00\bot)$, the environment is in local state 0, and all the other agents *are in local state* $(00\bot)$*. The abstract state* $\gamma' = \big((1,0,\bot)^1, \{(11\bot),(12\bot)\}, 1\big)$ *represents any concrete state in which agent 1 is in local state* $(00\bot)$*, the environment is in local state* 1*, and the template local states for all the other agents are* $(11\bot)$ *and* $(12\bot)$*. The abstract transition* $(\gamma, (n_-, move, \Xi_1), \gamma')$ *represents any concrete transition from a state represented by* γ *to a state represented by* γ'*, where agent 1 performs the action* n_-*, the environment performs the action* $move$*, and the actions for all the other agents are* $n_=$ *and* n_+*.*

Correspondence between a concrete system and the abstract model

We now establish a correspondence between a concrete system $\mathcal{S}(\overline{n})$ with $\overline{n} > \overline{m}$ and $\hat{\mathcal{S}}(\overline{m})$ as follows. Define $\delta_{\overline{n}} : G(\overline{n}) \to \hat{G}(\overline{m})$ to map concrete states from $\mathcal{S}(\overline{n})$ to abstract states in $\hat{\mathcal{S}}(\overline{m})$:

$$\delta_{\overline{n}}(g) = ((\delta_{\overline{n}}(g)).c, (\delta_{\overline{n}}(g)).\hat{a}) \,,$$

where

$$(\delta_{\overline{n}}(g)).c = \Big(ls_1^1(g), \ldots, ls_k^{\overline{m}.k}(g), ls_E(g)\Big)$$

is the concrete component and

$$\delta_{\overline{n}}(g)).\hat{a} = \Big(\Big\{tls_1^j(g) \colon j \in \{\overline{m}.1+1, \ldots, \overline{n}.1\}\Big\}, \ldots, \Big\{tls_k^j(g) \colon j \in \{\overline{m}.k+1, \ldots, \overline{n}.k\}\Big\}\Big)$$

is the abstract component. Assume a concrete transition $(g, \overline{a}, g')$ from state g to state g' by means of the joint action $\overline{a}$. For each agent template $\mathcal{T}_i$, consider $\zeta_{\overline{n}}.i$ to map $(g, \overline{a}, g')$ to a set of labels from Λ_i as follows: $(l, a, A, l') \in \zeta_{\overline{n}}.i(g, \overline{a}, g')$ iff the following hold:

- there is a $j \in \{\overline{m}.i+1, \ldots, \overline{n}.i\}$ such that $tls_i^j(g) = l$, $la_i^j(\overline{a}) = a$, and $tls_i^j(g') = l'$;
- $A = \Big\{la_{i'}^j(\overline{a}) \colon (i', j) \in \mathcal{A}(\overline{n})\Big\}$.

Finally, define $\theta_{\overline{n}}$ to map paths in $\mathcal{S}(\overline{n})$ to paths in $\hat{\mathcal{S}}(\overline{m})$ by

$$\theta_{\overline{n}}(g^1\overline{a}^1g^2\ldots) = \delta_{\overline{n}}(g^1)\left(\zeta_{\overline{n}}.1(g^1, \overline{a}^1, g^2), \ldots, \zeta_{\overline{n}}.k(g^1, \overline{a}^1, g^2)\right)\delta_n(g^2)\ldots.$$

By means of the above mappings we obtain the following simulation relation between $\mathcal{S}(\overline{n})$ and $\hat{\mathcal{S}}(\overline{m})$.

Lemma 4.1. *Let $\mathcal{S}(\overline{n})$ be a concrete system with $\overline{n} > \overline{m}$. Assume the relation $\hat{\sim}_s = \{(g, \gamma) \colon \delta_{\overline{n}}(g) = \gamma\}$ between $\mathcal{S}(\overline{n})$ and $\hat{\mathcal{S}}(\overline{m})$. Then, the following hold:*

1. *for every concrete initial state $g \in I(\overline{n})$, there is an initial abstract state $\gamma \in \hat{I}(\overline{m})$ with $(g, \gamma) \in \hat{\sim}_s$.*
2. *if $(g, \gamma) \in \hat{\sim}_s$ and $(g, g') \in R(\overline{n})$, then $(\gamma, \Xi_1, \ldots, \Xi_k, \gamma') \in \hat{R}(\overline{m})$ for some labels $\Xi_1, \ldots, \Xi_k$ and some abstract state γ' with $(g', \gamma') \in \hat{\sim}_s$.*

Proof.

1. $\delta_{\overline{n}}(g) \in \hat{I}(\overline{m})$;
2. Assume (g, g') by means of a joint action $\overline{a}$. Then,

$$\left(\delta_{\overline{n}}(g), \left(\zeta_{\overline{n}}.1(g, \overline{a}, g'), \ldots, \zeta_{\overline{n}}.k(g, \overline{a}, g')\right), \delta_{\overline{n}}(g')\right) \in \hat{R}(\overline{m}).$$

□

The above gives the correspondence between $\hat{\mathcal{S}}(\overline{m})$ and a concrete system $\mathcal{S}(\overline{n})$: every behaviour admitted by $\mathcal{S}(\overline{n})$ is representable by $\hat{\mathcal{S}}(\overline{m})$. This means that the satisfaction of a formula on the abstract model entails the satisfaction of the formula on every concrete system. However, it is not generally the case that every behaviour admitted by $\hat{\mathcal{S}}(\overline{m})$ has a concrete counterpart. This means that if a formula is not satisfied on the abstract model, then no conclusions can be drawn on the existence of a concrete system falsifying the formula. Indeed, $\hat{\mathcal{S}}(\overline{m})$ may contain spurious paths that correspond to the infinite repetition of certain cycles. Although a concrete system can perform these cycles, it can do so only for a finite number of repetitions. The exact number of repetitions depends on the size of the system. The following gives an example of a spurious path.

Example 4.3. *Consider the PIS $\mathcal{S} = ((L, Act, P, t), \mathcal{I}, \mathcal{V})$ built from one agent template, where*

- *$L = \{\iota, l, l'\}$;*
- *$Act = \{a, b\}$;*
- *$P(\iota) = \{a, b\}$, $P(l) = \{b\}$, $P(l') = \{a\}$;*
- *$t(\iota, a, \{a, b\}) = l$, $t(\iota, b, \{a, b\}) = l'$, $t(l, b, \{a, b\}) = l$, $t(l', a, \{a, b\}) = \iota$;*
- *$\mathcal{I} = \{\iota\}$;*
- *The definition of $\mathcal{V}$ is irrelevant for the purpose of this example.*

$\mathcal{S}$ is depicted in Figure 4.3a. A fragment of its abstract model $\hat{\mathcal{S}}(1)$ is depicted in Figure 4.3b. Observe the infinite abstract path

$$\left((l', \{l, l'\})\, \Xi_1\, (\iota, \{\iota, l\})\, \Xi_2 (l', \{l, l'\})\right)^{\omega}$$

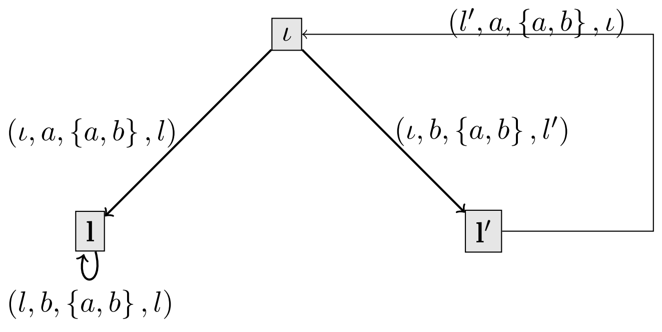
where:

$$\Xi_1 = \left\{(l', a, \{a, b\}, \iota), (l, b, \{a, b\}, l)\right\};$$
$$\Xi_2 = \left\{(\iota, a, \{a, b\}, l), (\iota, b, \{a, b\}, l'), (l, b, \{a, b\}, b)\right\}.$$

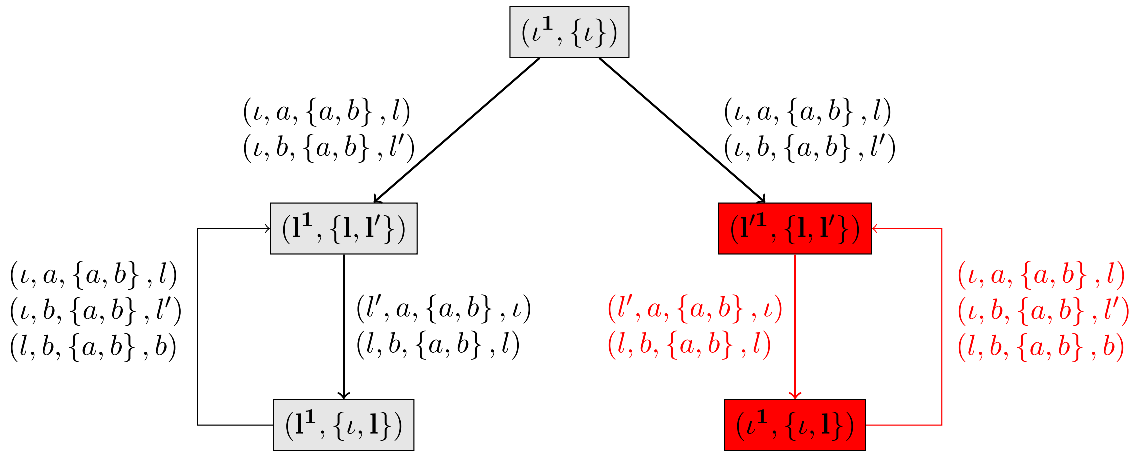
The above path represents a situation in which concrete agent 1 goes through the cycle of states (l', ι, l') infinitely often. However, for each repetition of the cycle, agent 1 may transition from state ι to state l' (by performing the action b) only if there is another agent that performs the action a at that round. If so, then the latter agent goes from state ι to state l in which it forever remains. At state l, the agent is not permitted by its protocol to perform the a action. It follows that agent 1 can only go through the cycle of states (ι, l', ι) a finite number of times. This number depends on the number of agents in the system.

4.2.3 Step 2: pruning

We now proceed to "eliminate" the spurious paths of the abstract model. We do this by building the computation forest of $\hat{\mathcal{S}}(\overline{m})$. The forest includes all and only the paths of $\hat{\mathcal{S}}(\overline{m})$ that do



(a) Agent template.


$(\iota^1, \{\iota\})$
$(\iota, a, \{a,b\}, l)$
$(\iota, b, \{a,b\}, l')$
$(\iota, a, \{a,b\}, l)$
$(\iota, b, \{a,b\}, l')$
$(\mathbf{l}^1, \{\mathbf{l}, \mathbf{l}'\})$
$(\mathbf{l}'^1, \{\mathbf{l}, \mathbf{l}'\})$
$(\iota, a, \{a,b\}, l)$
$(\iota, b, \{a,b\}, l')$
$(l, b, \{a,b\}, b)$
$(l', a, \{a,b\}, \iota)$
$(l, b, \{a,b\}, l)$
$(l', a, \{a,b\}, \iota)$
$(l, b, \{a,b\}, l)$
$(\iota, a, \{a,b\}, l)$
$(\iota, b, \{a,b\}, l')$
$(l, b, \{a,b\}, b)$
$(\mathbf{l}^1, \{\iota, \mathbf{l}\})$
$(\iota^1, \{\iota, \mathbf{l}\})$

(b) Fragment of the abstract model.

Figure 4.3: The abstraction of a PIS may contain spurious paths.

not contain a cycle with more than $td(\forall_{\overline{v}}\phi(\overline{v}))+1$ repetitions. Roughly speaking, following the discussion in Section 3.4, the forest contains all finite repetitions of a given cycle that may alter the satisfaction status of the formula to check. Note, however, that the forest cannot be used to verify the formula since it eliminates all infinite paths of $\hat{\mathcal{S}}(\overline{m})$; some of them might not be spurious. Nevertheless, the forest can be used to compute a cutoff, as we describe in the next section.

The computation forest of $\hat{\mathcal{S}}(\overline{m})$ is the disjoint union of the computation trees generated for each initial state in $\hat{\mathcal{S}}(\overline{m})$. For a state $\gamma \in \hat{G}(m)$ and $d \geq 1$, we associate a labelled computation tree $\mathfrak{T}(\gamma, d)$ rooted at $\dot{\gamma}$[2] that is inductively defined as follows.

Definition 4.5 (Computation tree)**.** *Given an abstract state γ in $\hat{\mathcal{S}}(\overline{m})$ and $d \geq 1$, the labelled computation tree $\mathfrak{T}(\gamma, d)$ rooted at $\dot{\gamma}$ is inductively defined as follows.*

- *$\mathfrak{T}^0(\gamma, d)$ consists of exactly the node $\dot{\gamma}$.*
- *$\mathfrak{T}^{n+1}(\gamma, d)$ consists of root node $\dot{\gamma}$, and for every transition $(\gamma, \Xi_1, \ldots, \Xi_k, \gamma') \in \hat{R}(\overline{m})$, $\dot{\gamma}$ has a subtree $\mathfrak{T}^n(\gamma', d)$ with edge label $\Xi_1, \ldots, \Xi_k$ iff there is no path in $\mathfrak{T}^{n+1}(\gamma, d)$ containing a cyclic repetition for more than d times.*

It is easy to see that there is a $q \in \mathbb{N}$ such that $\mathfrak{T}^q(\gamma, d)$ is isomorphic to every $\mathfrak{T}^{q'}(\gamma, d)$ with $q' \geq q$. We write $\mathfrak{T}(\gamma, d)$ to denote this tree. Note that $\mathfrak{T}(\gamma, d)$ is finite. A path $\mathfrak{b}$ in $\mathfrak{T}(\gamma, d)$ is a sequence $\dot{\gamma}^1(\Xi_1^1, \ldots, \Xi_k^1) \ldots (\Xi_1^{x-1}, \ldots, \Xi_k^{x-1})\dot{\gamma}^x$ such that $\dot{\gamma}^1$ is the root, $\dot{\gamma}^x$ is a leaf, and for every $1 \leq i \leq x-1$, $(\Xi_1^i, \ldots, \Xi_k^i)$ is the label for the edge between $\dot{\gamma}^i$ and $\dot{\gamma}^{i+1}$. We write $\mathfrak{b}(i)$ for the i-th node in $\mathfrak{b}$, $\mathfrak{b}(i, \Lambda_j)$ for the j-th component of the i-th label in $\mathfrak{b}$, $\mathfrak{b}[i]$ for the suffix $\dot{\gamma}^i(\Xi_1^i, \ldots, \Xi_k^i) \ldots$ of $\mathfrak{b}$, and $|\mathfrak{b}|$ for the number of nodes in $\mathfrak{b}$.

We can now define the computation forest of $\hat{\mathcal{S}}(\overline{m})$ with respect to $d \geq 1$.

Definition 4.6 (Computation forest)**.** *Given $d \geq 1$, the computation forest $\dot{\mathcal{S}}(\overline{m}, d)$ of $\hat{\mathcal{S}}(\overline{m})$ is the disjoint union $\dot{\mathcal{S}}(\overline{m}, d) = \amalg_{\gamma \in \hat{I}(\overline{m})} \mathfrak{T}(\gamma, d)$ of the computation trees generated for each initial state in $\hat{\mathcal{S}}(\overline{m})$.*

[2]We use $\dot{\gamma}$ to indicate the node $\dot{\gamma}$ in the computation tree that corresponds to the abstract state γ.

4.2.4 Step 3: counting

In the final step, the `PIS` procedure identifies a cutoff for $\mathcal{S}$ and $\forall_{\overline{v}}\phi(\overline{v})$. The cutoff is identified by computing the sufficient number of agents in a concrete system so that the system can simulate $\dot{\mathcal{S}}(\overline{m}, td(\forall_{\overline{v}}\phi(\overline{v})) + 1)$. Roughly speaking, if a concrete system simulates $\dot{\mathcal{S}}(\overline{m}, td(\forall_{\overline{v}}\phi(\overline{v})) + 1)$, it then admits all finite repetitions of a given cycle that may alter the satisfaction status of $\forall_{\overline{v}}\phi(\overline{v})$. Additionally, differently from $\dot{\mathcal{S}}(\overline{m}, td(\forall_{\overline{v}}\phi(\overline{v})) + 1)$, the concrete system admits all infinite paths of any bigger system. Thus, formally speaking, if a concrete system simulates $\dot{\mathcal{S}}(\overline{m}, td(\forall_{\overline{v}}\phi(\overline{v})) + 1)$, then the concrete system is cycle-stuttering simulation equivalent up to degree $td(\forall_{\overline{v}}\phi(\overline{v})) + 1$ to every bigger system. We prove this in section 4.2.6 where we show that said concrete system is a cutoff system. In this section, we show that cutoffs always exist.

We begin by considering a path $\mathfrak{b}$ in the computation tree $\mathfrak{T}(\gamma, d)$, for an abstract state γ and $d \geq 1$. We define $\mathfrak{b}(i, \Lambda_j) = \emptyset$ whenever $\mathfrak{b}(i)$ is a leaf node. Then, given a template state $l \in L_j$ of agent template $\mathcal{T}_j$, we define

$$out_j(\mathfrak{b}(i), l) = \left|\left\{a \colon (l, a, A, l') \in \mathfrak{b}(i, \Lambda_j)\right\}\right|.$$

In any concrete transition (g, g') that corresponds to $(\mathfrak{b}(i-1), (\Xi_1, \ldots, \Xi_k), \mathfrak{b}(i))$, $out_j(\mathfrak{b}(i-1), l)$ is the sufficient number of concrete agents from $\mathcal{T}_j$ that need to be in l in g for the transition to be enabled. For any concrete path π that corresponds to $\mathfrak{b}$, we let $cutoff_j(\mathfrak{b}, l)$ be the sufficient number of concrete agents from $\mathcal{T}_j$ that need to be in l in $\pi(1)$ for the concrete path to be performed. $cutoff_j(\mathfrak{b}, l)$ is defined as follows:

$$cutoff_j(\mathfrak{b}, l) = out_j(\mathfrak{b}(1), l) + \sum_{(l' : (l, a, A, l') \in \mathfrak{b}(1, \Lambda_j))} cutoff_j(\mathfrak{b}[1], l').$$

For any concrete state g that corresponds to a node $\dot{\gamma}$ in $\dot{\mathcal{S}}(\overline{m}, d)$, we consider $cutoff_j(\gamma, l)$ to be the sufficient number of concrete agents from $\mathcal{T}_j$ that need to be in l in g so that any concrete path originating from g and corresponding to a path in $\Pi(\dot{\gamma})$ can be performed. $cutoff_j(\gamma, l)$ is defined as follows:

$$\mathit{cutoff}_j(\dot{\gamma}, l) = \max(\mathit{cutoff}_j(\mathfrak{b}, l) \mid \mathfrak{b} \text{ is a path originating from } \dot{\gamma}).$$

We now define the sufficient number $\mathit{cutoff}_j(\dot{\gamma})$ of concrete agents from $\mathcal{T}_j$ that need to be in g so that any concrete path originating from g and corresponding to a path in $\Pi(\dot{\gamma})$ can be performed.

$$\mathit{cutoff}_j(\dot{\gamma}) = \sum_{l \in L_j} \mathit{cutoff}_j(\dot{\gamma}, l)$$

Finally, we let $\mathit{cutoff}_j(\dot{\mathcal{S}}(\overline{m}, d))$ be the sufficient number of concrete agents from $\mathcal{T}_j$ so that a concrete system can simulate every path in $\dot{\mathcal{S}}(\overline{m}, d)$. $\mathit{cutoff}_j(\dot{\mathcal{S}}(\overline{m}, d))$ is given by the following:

$$\mathit{cutoff}_j(\dot{\mathcal{S}}(\overline{m}, d)) = \max(\mathit{cutoff}_j(\dot{\gamma}) \mid \dot{\gamma} \text{ is a root node }) + \overline{m}.j$$

We can now define the *cutoff function* $\mathit{cutoff}(\mathcal{S}, \overline{m}, d)$. The function takes as input a PIS $\mathcal{S}$ of k roles, $\overline{m} \in \mathbb{N}^k$, $d \geq 1$, and returns the cutoff for $\mathcal{S}$ and all $\overline{m}$-indexed formulae with temporal depth less than d.

Definition 4.7 (Cutoff function). *Given a PIS $\mathcal{S}$ of k roles, $\overline{m} \in \mathbb{N}^k$, and $d \geq 1$, the* cutoff function *$\mathit{cutoff}(\mathcal{S}, \overline{m}, d)$ is defined as follows:*

$$\mathit{cutoff}(\mathcal{S}, \overline{m}, d) = \Big(\mathit{cutoff}_1(\dot{\mathcal{S}}(\overline{m}, d)), \ldots, \mathit{cutoff}_k(\dot{\mathcal{S}}(\overline{m}, d))\Big)$$

The cutoff system $\mathcal{S}(\mathit{cutoff}(\mathcal{S}, \overline{m}, d))$ is related to $\dot{\mathcal{S}}(\overline{m}, d)$ by means of the following simulation relation.

Lemma 4.2. *Let $\overline{c} = \mathit{cutoff}(\mathcal{S}, \overline{m}, d)$. Consider $\dot{\sim}_s$ to be a relation between the nodes in $\dot{\mathcal{S}}(\overline{m}, d)$ and the states in $\mathcal{S}(\overline{c})$ that is defined as follows: $(\dot{\gamma}, g) \in \dot{\sim}_s$ iff*

- *$\delta_{\overline{c}}(g) = \gamma$;*
- *for each agent template $\mathcal{T}_i$ and for each template state $l \in L_i$, we have that*

$$\Big|\Big\{(i, j) \colon j \in \{\overline{m}.i + 1, \ldots, \overline{c}.i\} \text{ and } \mathit{tls}_i^j(g) = l\Big\}\Big| \geq \mathit{cutoff}_i(\dot{\gamma}, l).$$

Then, the following hold:

1. *for every root node $\dot\gamma$ in $\dot{\mathcal{S}}(\overline{m}, d)$, there is an initial concrete state $g \in I(\overline{c})$ with $(\dot\gamma, g) \in \dot\sim_s$;*

2. *if $(\dot\gamma, g) \in \dot\sim_s$ and $\dot\gamma'$ is a child of $\dot\gamma$, then $(\dot\gamma', g') \in \dot\sim_s$ for some $g' \in G(\overline{c})$ with $(g, g') \in R(\overline{c})$.*

Proof.

1. By the definition of $\mathit{cutoff}(\mathcal{S}, \overline{m}, d)$.

2. Suppose that $(\dot\gamma, g) \in \dot\sim_s$. Let $\dot\gamma'$ be a child of $\dot\gamma$. Let $(\Xi_1, \ldots, \Xi_k)$ be the label for the edge between $\dot\gamma$ and $\dot\gamma'$, and assume an arbitrary template state l of agent template $\mathcal{T}_i$. From $(\dot\gamma, g) \in \dot\sim_s$ and by the definition of $\mathit{cutoff}_i(\gamma, l)$, the following holds:

$$\left|\left\{(i,j) \colon j \in \{\overline{m}.i+1, \ldots, \overline{c}.i\} \text{ and } tls_i^j(g) = l\right\}\right| \geq \sum_{(l' : (l,a,A,l') \in \Xi_i)} \mathit{cutoff}_i(\gamma, l').$$

Therefore, there is a sufficient number of agents in each template state l in g to appropriately populate each template state l' that results from l in a template transition. It follows that there is a global state g' with $(g, g') \in R(\overline{c})$ and $(\dot\gamma', g') \in \dot\sim_s$.

□

We write $\dot{\mathcal{S}}(\overline{m}, d) \dot\leq_s \mathcal{S}(\overline{c})$ to indicate that $\mathcal{S}(\overline{c})$ simulates $\dot{\mathcal{S}}(\overline{m}, d)$ by means of the above relation.

4.2.5 Summary

The `PIS` procedure is given by Algorithm 1. Given a PIS $\mathcal{S}$ an $\overline{m}$-indexed ACTLK formula $\forall_{\overline{v}}\phi(\overline{v})$, the procedure constructs the abstract model $\hat{\mathcal{S}}(\overline{m})$. It then builds the computation forest $\dot{\mathcal{S}}(\overline{m}, td(\phi)+1)$ of $\hat{\mathcal{S}}(\overline{m})$. $\dot{\mathcal{S}}(\overline{m}, td(\phi)+1)$ admits all paths in $\hat{\mathcal{S}}(\overline{m})$ that do not contain a cycle with more than $td(\phi)+1$ repetitions. With the construction of $\dot{\mathcal{S}}(\overline{m}, td(\phi)+1)$, the procedure iteratively checks for a concrete system $\mathcal{S}(\overline{c})$ such that $\mathcal{S}(\overline{c})$ simulates $\dot{\mathcal{S}}(\overline{m}, td(\phi)+1)$. $\mathcal{S}(\overline{c})$ is the cutoff system. Following the cutoff identification, `PIS` checks the set $\{\mathcal{S}(\overline{x}) \colon \overline{m} \leq \overline{x} \leq \overline{c}\}$ of

Algorithm 1 Parameterised model checking procedure for parameterised interpreted systems.

```
1:  procedure PIS($\mathcal{S}$,$\forall_{\overline{v}}\phi(\overline{v})$)
2:      build $\hat{\mathcal{S}}(\overline{m})$
3:      build $\dot{\mathcal{S}}(\overline{m}, td(\forall_{\overline{v}}\phi(\overline{v})) + 1)$
4:      $\bar{c} \leftarrow \bar{m}$
5:      do
6:          increase $\bar{c}$ in a breadth-first manner
7:      until $\dot{\mathcal{S}}(\overline{m}, d) \dot{\leq}_s \mathcal{S}(\bar{c})$
8:      for all $\bar{x}$ with $\bar{m} \leq \bar{x} \leq \bar{c}$ do
9:          if $\mathcal{S}(\bar{x}) \not\models \phi[trivial]$ then
10:             return $false$;
11:         end if
12:     end for
13:     return $true$;
14: end procedure
```

concrete systems against the trivial instantiation $\phi[trivial]$ of $\forall_{\overline{v}}\phi(\overline{v})$. If $\phi[trivial]$ is not satisfied by at least one system, then the procedure returns $false$, otherwise it returns $true$.

This concludes the description of the PIS procedure. Algorithm 1 provides a methodology for solving the parameterised model checking problem by checking all systems up to the cutoff system. Note that, by Lemma 4.2, a cutoff system can always be identified. Therefore, PIS is a complete procedure. We assess the soundness of PIS in the next section.

4.2.6 Proof of soundness

Theorem 4.1. *Let $\mathcal{S}$ be a parameterised interpreted system and $\forall_{\overline{v}}\phi(\overline{v})$ be an $\overline{m}$-indexed ACTLK formula. Then, PIS$(\mathcal{S}, \forall_{\overline{v}}\phi(\overline{v}))$ returns $true$ iff $\mathcal{S} \models \forall_{\overline{v}}\phi(\overline{v})$.*

It suffices to show that if $\dot{\mathcal{S}}(\overline{m}, td(\forall_{\overline{v}}\phi(\overline{v}))+1) \dot{\leq} \mathcal{S}(\overline{c})$, then $\mathcal{S}(\overline{c})$ is a cutoff system. The proof is given in two parts. In the first part, we show that $\mathcal{S}(\overline{c})$ cycle-stuttering simulates up to degree $td(\forall_{\overline{v}}\phi(\overline{v})) + 1$ every bigger system. By Theorem 3.2, we then get the following:

$$\mathcal{S}(\overline{c}) \models \forall_{\overline{v}}\phi(\overline{v}) \text{ implies } \mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v}), \text{ for any } \overline{n} \geq \overline{c} \tag{1}$$

In the second part, we show that every bigger system cycle-stuttering simulates $\mathcal{S}(\overline{c})$ up to degree $td(\forall_{\overline{v}}\phi(\overline{v})) + 1$. By Theorem 3.2, we then obtain the following:

$$\mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v}) \text{ implies } \mathcal{S}(\overline{c}) \models \forall_{\overline{v}}\phi(\overline{v}), \text{ for any } \overline{n} \geq \overline{c} \tag{2}$$

From (1) and (2) it follows that

$$\forall \overline{m} \leq \overline{x} \leq \overline{c}.\, \mathcal{S}(\overline{x}) \models \forall_{\overline{v}}\phi(\overline{v}) \text{ iff } \forall \overline{n} \geq \overline{c}.\, \mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v})$$

Therefore, $\mathcal{S}(\overline{c})$ is a cutoff system.

Part A: the cutoff system cycle-stuttering simulates every bigger system

Lemma 4.3. *Let $\mathcal{S}$ be a parameterised interpreted system and $\forall_{\overline{v}}\phi(\overline{v})$ be an $\overline{m}$-indexed ACTLK formula. Then, $\mathcal{S}(\overline{n}) \leq_{(td(\forall_{\overline{v}}\phi(\overline{v}))+1).css} \mathcal{S}(\overline{c})$, where $\dot{\mathcal{S}}(\overline{m}, td(\forall_{\overline{v}}\phi(\overline{v})) + 1) \lesssim \mathcal{S}(\overline{c})$ and $\overline{n} \geq \overline{c}$.*

For ease of explanation and for brevity of the proof, we introduce the following operation on a given path.

Definition 4.8. *Let $\sigma = C_1\mathfrak{C}_1^{x_1} \dots C_z\mathfrak{C}_z^{x_z}$ be a cyclic decomposition of a path. Given a natural number d, the d-destuttering $d.ds(\sigma) = d.ds(C_1)d.ds(\mathfrak{C}_1^{x_1}) \dots d.ds(C_z)d.ds(\mathfrak{C}_z^{x_z})$ of σ is defined as follows:*

- *$d.ds(C_i) = C_i$ for a non-cyclic block C_i;*
- *$d.ds(\mathfrak{C}_i^{x_i}) = (d.ds(\mathfrak{C}_i))^{x_i'}$, where $x_i' = \min(d, x_i)$, for a cyclic block $\mathfrak{C}_i^{x_i}$.*

In other words, $d.ds(\sigma)$ includes up to d the repetitions of each cyclic block in σ.

Proof. Let $td(\forall_{\overline{v}}\phi(\overline{v})) + 1 = d$ and fix $\overline{n} \geq \overline{c}$. We show that $\mathcal{S}(\overline{n}) \leq_{d.css} \mathcal{S}(\overline{c})$. We define the relations $Q^0, Q^1, \dots, Q^d$ between $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{c})$. We then show that Q^d is a cycle-stuttering simulation relation of degree d between $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{c})$.

We start by defining the relations $Z^0, Z^1, \dots, Z^d$ between the states in $\mathcal{S}(\overline{n})$ and the nodes in $\dot{\mathcal{S}}(\overline{m}, d+1)$. A state $g \in G(\overline{n})$ and a node $\dot{\gamma}$ in $\dot{\mathcal{S}}(\overline{m}, d)$ are Z^0-related if $\delta_{\overline{n}}(g) = \gamma$. For any $0 < i \leq d$, a state $g \in G(\overline{n})$ and a node $\dot{\gamma}$ in $\dot{\mathcal{S}}(\overline{m}, d)$ are Z^i-related if for every finite path $\pi \in \Pi(g)$ there is a path $\mathfrak{b} \in \Pi(\dot{\gamma})$, a cyclic decomposition σ of $\theta_{\overline{n}}(\pi)$ and a cyclic decomposition σ' of $\mathfrak{b}$ such that $\sigma' = (i-1).ds(\sigma)$.

Finally, for all $0 \leq i \leq d$, we define two states g and g' in $G(\overline{n})$ and $G(\overline{c})$, respectively, to be Q^i-related if there is a node $\dot{\gamma}$ in $\dot{\mathcal{S}}(\overline{m}, d)$ such that $(g, \dot{\gamma}) \in Z^i$ and $(\dot{\gamma}, g') \in \dot{\sim}_s$.

We show that Q^d is a cycle-stuttering simulation of degree d between $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{c})$. We do this by induction on d.

For the base step, we have that $d = 0$. Assume an initial state $g \in I(\overline{n})$. We have to show that there is an initial state $g' \in I(\overline{c})$ with $(g, g') \in Q^0$. Let $\delta_{\overline{n}}(g) = \gamma$. We have that $\dot{\gamma}$ is a root node in $\dot{\mathcal{S}}(\overline{m}, d)$. Therefore, $(g, \dot{\gamma}) \in Z^0$. By Lemma 4.2, there is an initial state $g' \in I(\overline{c})$ with $(\dot{\gamma}, g') \in \dot{\sim}_s$. Hence, $(g, g') \in Q^0$. Let $(g, g') \in Q^0$ be an arbitrary pair of states. We have to show the following:

1. $V(\overline{n})(g) = V(\overline{c})(g')$.

   Since $(\dot{\gamma}, g') \in \dot{\sim}_s$, $\delta_{\overline{c}}(g') = \gamma = \delta_{\overline{n}}(g)$. As the concrete component in γ is built from the local states of the agents $\mathcal{A}(\overline{m})$, it follows that $V(\overline{n})(g) = V(\overline{c})(g')$.

2. if $\mathcal{K}_i^j(g, g^1)$, for $(i, j) \in \mathcal{A}(\overline{m})$, then $\mathcal{K}'^j_i(g', g'^1)$ for some g'^1 with $(g^1, g'^1) \in Q^0$.

   Assume that $\mathcal{K}_i^j(g, g^1)$, for some agent $(i, j) \in \mathcal{A}(\overline{m})$. By construction of $\dot{\mathcal{S}}(\overline{m}, d)$ there is a node $\dot{\gamma}$ with $\delta_{\overline{n}}(g^1) = \gamma$. By Lemma 4.2, there is a state $g'^1 \in G(\overline{c})$ with $\delta_{\overline{c}}(g'^1) = \gamma$. Therefore, $\mathcal{K}'^j_i(g', g'^1)$ and $(g^1, g'^1) \in Q^0$.

So, the claim is true for $d = 0$. Assume that for all $1 \leq x < d$, Q^x is a cycle-stuttering simulation of degree x between $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{c})$. We show that this is the case for $x = d$.

Assume an initial state $g \in I(\overline{n})$. We have to show that there is an initial state $g' \in I(\overline{c})$ with $(g, g') \in Q^x$. Let $\delta_{\overline{n}}(g) = \gamma$. We have that $\dot{\gamma}$ is a root node in $\dot{\mathcal{S}}(\overline{m}, x)$. By Lemma 4.2, there is an initial state $g' \in I(\overline{c})$ with $(\dot{\gamma}, g') \in \dot{\sim}_s$. We prove that $(g, g') \in Q^x$. That is, we prove the following:

1. $(g, \dot{\gamma}) \in Z^x$.

Let $\pi \in \Pi(g)$ be a finite path. By Lemma 4.1, $\theta_{\overline{n}}(\pi) \in \Pi(\gamma)$. By construction of $\dot{\mathcal{S}}(\overline{m}, x)$, $\Pi(\dot{\gamma})$ contains all paths in $\Pi(\gamma)$ that do not have a cyclic repetition for more than x times. It follows that there is a path $\mathfrak{b} \in \Pi(\dot{\gamma})$, a cyclic decomposition σ of $\theta_{\overline{n}}(\pi)$ and a cyclic decomposition σ' of $\mathfrak{b}$ such that $\sigma' = (x-1).ds(\sigma)$. Thus, $(g, \dot{\gamma}) \in Z^x$.

2. $(\dot{\gamma}, g') \in \dot{\sim}_s$.

   Given above.

Let $(g, g') \in Q^x$ be an arbitrary pair of states. We have to show that for every path $\pi \in \Pi(g)$, there is a path $\pi' \in \Pi(g')$, a cyclic decomposition $B_1 B_2 \ldots B_z$ of π, and a cyclic decomposition $B'_1 B'_2 \ldots B'_z$ of π' such that for all $i \geq 1$, $B_i \propto_{x-1} B'_i$.

Let $\pi \in \Pi(g)$. Since the transition relation of $\mathcal{S}(\overline{n})$ is serial, we have that π is an infinite path. That is, π is of the form ρb^ω, where ρ is a finite prefix of π and b is a cycle on states. We construct a path $\pi'' \in \Pi(g')$ as required. By definition of Q^x, there is a node $\dot{\gamma}$ in $\dot{\mathcal{S}}(\overline{m}, x)$ with $(g, \dot{\gamma}) \in Z^x$ and $(\dot{\gamma}, g') \in \dot{\sim}_s$. By $(g, \dot{\gamma}) \in Z^x$, there is a path $\mathfrak{b} \in \Pi(\dot{\gamma})$, a cyclic decomposition σ of $\theta_{\overline{n}}(\rho b^1)$, and a cyclic decomposition σ' of $\mathfrak{b}$ such that $\sigma' = (x-1).ds(\sigma)$. By $(\dot{\gamma}, g') \in \dot{\sim}_s$, there is a path $\pi' \in \Pi(g')$ such that $(\mathfrak{b}(i), \pi(i)) \in \dot{\sim}_s$ for every $1 \leq i \leq |\mathfrak{b}|$. Therefore π' is of the form $\rho' b'$, where b' is a cycle with $\theta_{\overline{c}}(b') = \theta_{\overline{n}}(b)$. Let $\pi'' = \rho' b'^\omega$. It should be clear that π'' is a valid path in $\mathcal{S}(\overline{c})$. We show that π'' is as required by cycle-stuttering simulation.

Consider ψ to be the cyclic decomposition σ, but with each abstract state $(\theta_{\overline{n}}(\pi))(i)$ appearing in σ replaced by the corresponding concrete state $\pi(i)$; in other words, σ' expresses the application of σ to π. Consider ψ' to be the cyclic decomposition σ', but with each node $(\mathfrak{b})(i)$ appearing in σ' replaced by the corresponding concrete state $\pi'(i)$. Now define $\psi^1 = \psi B$ and $\psi'^1 = \psi' B'$, where $B = b^\omega$ and $B' = b'^\omega$ are cyclic blocks.

We prove that ψ^1, ψ'^1 are the required cyclic decompositions of π, π'', respectively, of cycle-stuttering simulation. Since $\psi' = (x-1).ds(\psi)$, it suffices to show that for each pair C, C' of corresponding non-cyclic blocks appearing in ψ^1, ψ'^1, respectively, we have that $(C(j), C'(j)) \in Q^{x'}$, for each $1 \leq j \leq |C|$, where x' is the maximum number in $\{1, \ldots, x-1\}$ such that both blocks can "see" ahead x' repetitions of every cyclic block they are composed into. To show this, let $j \geq 1$ and $\dot{\beta}$ be the j-th node appearing in the corresponding block of C in σ'. We have that $(\dot{\beta}, C'(j)) \in \dot{\sim}_s$. Also, by construction of $\dot{\mathcal{S}}(\overline{m}, x)$ and the definition of x', $\Pi(\dot{\beta})$ contains all abstract paths in $\Pi(\beta)$ that do not contain a cyclic repetition for more than x' times. As $\delta_{\overline{n}}(C(j)) = \beta$, the latter gives $(C(j), \dot{\beta}) \in Z^{x'}$. Hence, $(C(j), C'(j)) \in Q^{x'}$. By the inductive hypothesis, $Q^{x'}$ is a cycle stuttering simulation between $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{c})$ of degree x'.

We have thus proven that for every path $\pi \in \Pi(g)$, there is a path $\pi' \in \Pi(g')$, a cyclic decomposition $B_1 B_2 \ldots B_z$ of π, and a cyclic decomposition $B'_1 B'_2 \ldots B'_z$ of π' such that for all $i \geq 1$, $B_i \propto_{d-1} B'_i$. As (g, g') was arbitrary, we have shown that Q^d is a cycle-stuttering simulation relation between $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{c})$. That is, we have demonstrated that $\mathcal{S}(\overline{n}) \leq_{(td(\forall_{\overline{v}}\phi(\overline{v}))+1).css} \mathcal{S}(\overline{c})$. $\square$

Part B: every bigger system cycle-stuttering simulates the cutoff system

Lemma 4.4. *Let $\mathcal{S}$ be a parameterised interpreted system and $\forall_{\overline{v}}\phi(\overline{v})$ be an $\overline{m}$-indexed ACTLK formula. Then, $\mathcal{S}(\overline{c}) \leq_{(td(\forall_{\overline{v}}\phi(\overline{v}))+1).css} \mathcal{S}(\overline{n})$, where $\dot{\mathcal{S}}(\overline{m}, td(\forall_{\overline{v}}\phi(\overline{v})) + 1) \mathrel{\dot{\leq}} \mathcal{S}(\overline{c})$ and $\overline{n} \geq \overline{c}$.*

Proof. We prove a stronger result, i.e., that for every $\overline{n} \geq \overline{c}$, we have that $\mathcal{S}(\overline{n})$ simulates $\mathcal{S}(\overline{c})$ (see Section 2.3 for the properties of a simulation relation).

We start by defining the relation $Q \subseteq G(\overline{c}) \times G(\overline{n})$ between the states in $\mathcal{S}(\overline{c})$ and the states in $\mathcal{S}(\overline{n})$. Two states g, g' are Q-related if: (i) the states are projected into the same abstract state in $\hat{\mathcal{S}}(\overline{m})$; (ii) for each agent template $\mathcal{T}_i$, the agents $(i, \overline{c}.i + 1), \ldots, (i, \overline{n})$ are at the same template local state with agent $(i, 1)$ in g'. Formally, $(g, g') \in Q$ if the following holds:

$$\delta_{\overline{c}}(g) = \delta_{\overline{n}}(g') \text{ and for each template role } i \text{ and } \overline{c}.i + 1 \leq j \leq \overline{n}.i, tls_i^j(g') = tls_i^1(g').$$

We show that Q is a simulation relation between $\mathcal{S}(\overline{c})$ and $\mathcal{S}(\overline{n})$. It should be clear that for every initial state $g \in I(\overline{c})$, there is an initial state $g' \in I(\overline{n})$ with $(g, g') \in Q$. Let $(g, g') \in Q$ be an arbitrary pair of states. We have to show the following.

(i) $V(\overline{n})(g) = V(\overline{c})(g')$;

The requirement follows from $\delta_{\overline{n}}(g) = \delta_{\overline{c}}(g')$.

(ii) if $\mathcal{K}_i^j(g, g^1)$, for some agent $(i, j) \in \mathcal{A}(\overline{m})$, then $\mathcal{K'}_i^j(g', g'^1)$ for some g'^1 with $(g^1, g'^1) \in Q$.

Assume that $\mathcal{K}_i^j(g, g^1)$, for some $(i, j) \in \mathcal{A}(\overline{m})$. Consider a path $\iota\overline{a}^1 \dots \overline{a}^{x-1} g^x$ with ι being an initial state and $g^x = g^1$. For each $\overline{c}$-tuple of actions $\overline{a}^j$, define the extension of $\overline{a}^j$ to an $\overline{n}$-tuple $ex_{\overline{n}}(\overline{a}^j)$ of actions as follows:

- for each template role r and for each $1 \leq q \leq \overline{c}.r$, $la_r^q(ex_{\overline{n}}(\overline{a}^j)) = la_r^q(\overline{a}^j)$;
- for each template role r and for each $\overline{c}.r + 1 \leq q \leq \overline{n}.r$, $la_r^q(ex_{\overline{n}}(\overline{a}^j)) = la_r^1(\overline{a}^j)$.

In other words, $ex_{\overline{n}}(\overline{a}^j)$ extends $\overline{a}^j$ to an $\overline{n}$-tuple by copying the action of concrete agent $(i, 1)$, for each agent template i, in $\overline{a}^j$ for $\overline{n}.i - \overline{c}.i$ times. Let $\iota' \in G(\overline{n})$ be such that $(\iota, \iota') \in Q$. Consider the path

$$\iota' ex_{\overline{n}}(\overline{a}^1) \dots ex_{\overline{n}}(\overline{a}^{x-1}) g'^1$$

in $\mathcal{S}(\overline{n})$ obtained by performing the sequence of actions $ex_{\overline{n}}(\overline{a}^1), \dots, ex_{\overline{n}}(\overline{a}^{x-1})$. It should be clear that this is a valid path in $\mathcal{S}(\overline{n})$. We have that $\mathcal{K'}_i^j(g', g'^1)$ and $\delta_{\overline{n}}(g'^1) = \delta_{\overline{c}}(g^1)$. Therefore, $(g^1, g'^1) \in Q$.

(iii) if $(g, g^1) \in R(\overline{c})$ for some $g^1 \in G(\overline{c})$, then there is a $g'^1 \in G(\overline{n})$ such that $(g', g'^1) \in R(\overline{n})$ and $(g^1, g'^1) \in Q$;

Assume $(g, g^1) \in R(\overline{c})$ by means of a joint action $\overline{a}$. Then, the action $ex_{\overline{n}}(\overline{a})$ is enabled at g' resulting at a global state g'^1 with $(g^1, g'^1) \in Q$.

As (g, g') was arbitrary, we have shown that Q is a simulation relation between $\mathcal{S}(\overline{c})$ and $\mathcal{S}(\overline{n})$. It follows that that $\mathcal{S}(\overline{c}) \leq_{(td(\forall_{\overline{v}}\phi(\overline{v}))+1).css} \mathcal{S}(\overline{n})$.

□

4.3 Applications

4.3.1 Autonomous robot

In Section 3.1.1 we represented the autonomous robots scenario as a parameterised interpreted system $\mathcal{S}_{AR}$. We now consider the indexed ACTLK formula

$$\phi_{AR} = \forall_{\{v,u\}} AG\left((halted, v) \rightarrow K^{v}_{robot}\left(AXAX(halted, u)\right)\right)$$

expressing "for every robot v, whenever v is halted, it knows that in the next two time steps every other robot is halted".

We analyse this property using the `PIS` procedure. We first observe that ϕ_{AR} is a 2-indexed formula and has a temporal depth of 3. Therefore, in step 1, we build the abstract model $\hat{\mathcal{S}}_{AR}(2)$. A fragment of $\hat{\mathcal{S}}_{AR}(1)$ is depicted in Figure 4.2; $\hat{\mathcal{S}}_{AR}(2)$ is similarly obtained by considering a concrete component with two agents. In step 2, we build the pruned computation forest $\dot{\mathcal{S}}_{AR}(2,4)$. The forest does not include repetitions of a cycle for more than 4 times. In step 3, we find a concrete system that simulates $\dot{\mathcal{S}}_{AR}(2,4)$. From the description of $\mathcal{S}_{AR}$, we have that every robot halts whenever the value of its sensor is greater than or equal to 3. As a sensor reading may differ at most 1 position from the actual position of a robot, every robot halts by position 4. Therefore, if an abstract state γ represents position 4 for all the robots, then the abstract component $\gamma.\hat{a}$ of γ never changes in an abstract transition. From Figure 4.2, we can observe that for any position that is less than 4, each template state $(x, y, \bot)$ (expressing position x, sensor value y, and that the robot has not halted) in $\gamma.c$ may be updated in an abstract transition to either: the template states $(x+1, x, \bot), (x+1, x+1, \bot), (x+1, x+2, \bot)$ if $y < 3$; or to the template state $(x, y, \top)$ if $y \geq 3$. Therefore, for each round prior to round 4, at most 4 concrete agents can perform every template transition indicated by the abstract transition relation at the round. It follows that the concrete system built from $4 \times 4 + 2$ agents can simulate $\dot{\mathcal{S}}_{AR}(2,4)$.

We can thus assess the correctness of $\mathcal{S}_{AR}$ by checking each concrete system $\mathcal{S}_{AR}(2), \ldots, \mathcal{S}_{AR}(18)$ against the trivial instantiation $AG\left((halted, 1) \rightarrow K^{1}_{robot}\left(AXAX(halted, 2)\right)\right)$ of ϕ_{AR}. The latter queries can be put to any epistemic model checker, which would return $true$, thereby establishing the correctness of $\mathcal{S}_{AR}$ irrespectively of the number of robots present.

4.3.2 The Beta swarm aggregation algorithm

Swarm robotics concerns the coordination and analysis of an unbounded collection of behaviourally simple and identical robotic agents [BDT99, Şah05, ŞW08]. The interaction between the agents and their environment is designed to exhibit a collective, emergent behaviour often inspired by biological systems, e.g., ants [Eng06]. As justified in [Şah05], despite the lack of centralised coordination, biological swarm-based systems can still be robust, scalable, and flexible. It is therefore of interest to design swarm robotic systems that they can be shown to be in compliance with their specifications. To do this, we need to analyse the properties of a swarm irrespectively of the number of robots in the system.

This section applies the PIS procedure to the Beta aggregation algorithm [Nem05], a protocol used to aggregate robots somewhere on a grid. In line with common theoretical assumptions made in the literature [Şah05, BFBD13] we here assume that each agent operates on a two-dimensional arena and communicates with its peers and the environment by means of a wireless sensor of limited range. A robot is said to be in another robot's neighbourhood if the position of the former is in the range of the latter's sensor. We consider the arena to be finite and we allow it to wrap around, i.e., for an $\alpha \times \alpha$ arena, the cell $(1, 1)$ is one position to the right of the cell $(1, \alpha)$. We also assume that the robots update their state with high frequency; so we can model them by assuming synchronicity even if at any instance some robots may be idle [DWFZ12].

To begin, given a robot i, define a *lost robot* j to be a robot that was in the neighbourhood of i in the previous step but it is not in the current step. Each robot i obeys two simple rules. The first rule concerns the number of i's neighbours that can observe each of i's lost robots in their neighbourhoods. If this number is less than a predefined threshold β for at least one lost robot, then robot i performs a 180° turn. The second rule concerns the number of i's neighbours. If the first rule does not apply and the number of i's neighbours increases during a time step, then the robot performs a random 90° turn. If neither rule can be applied, the robot moves forward one cell.

Fix a 5×5 arena, assume a communication range of 1, and let $\beta = 0$. We now encode the Beta algorithm as a PIS $\mathcal{S}_\beta$ consisting of an agent template R_β that reflects the rules above for the action selection and state evolution. Figure 4.4 depicts a fragment of R_β. A state $((x, y), X, Y, Z, Q, d, p)$ encodes the following:

1. the position (x, y) of the robot in the arena;
2. the set X of occupied neighbouring cells;
3. the set Y of occupied neighbouring cells in the previous step;
4. the set Z of occupied cells in the robot's neighbours' neighbourhoods;
5. the set Q of cells occupied in the current step by a neighbour of the previous step;
6. the direction $d \in \{North, East, South, West\}$ of movement;
7. the phase $p \in \{U1, U2, M\}$ of the encoding.

Define $c1 := (Q \setminus X) \setminus Z \neq \emptyset$ and $c2 := |X| > |Y|$ to be two boolean conditions that may hold on a state. The condition $c1$ holds on states enabling the first rule above, whereas $c2$ holds on states enabling the second rule. To update the sets defining the conditions, the modelling assumes three alternating phases $p = U1, p = U2, p = M$ for each time step. The update for each phase is accomplished by appropriately associating actions with cells and sets of cells. The set of actions is given by $Act = Loc \cup (Loc \times \mathcal{P}(Loc))$, where $Loc = \{(x, y) \colon 1 \leq x \leq 5, 1 \leq y \leq 5\}$ is the set of cells in the arena. The Loc actions are used to update the X sets in phase $U1$ with every robot signaling its position, whereas the $\mathcal{P}(Loc)$ actions are used to update the Z sets in phase $U2$ with every robot signaling its set X of neighbours. In phase M, each robot moves to a cell (x, y), resulting from which rule is applicable in the step, by performing the action (x, y). In this phase, the Y and Q sets are also updated. More precisely, the protocol for a state $s = ((x, y), (X, Y, Z, Q, d, p))$ is defined as follows.

- For $(x', y') \in Loc$ and $p = U1$, we have $(x', y') \in P(s)$ iff $(x', y') = (x, y)$;
- For $X' \in \mathcal{P}(Loc)$, we have $X' \in P(s)$ iff $X' = X$ and $p = U2$.

$$\begin{tikzcd}
 & & \boxed{((\mathbf{3},\mathbf{3}),(\emptyset,\emptyset,\emptyset,\emptyset,\mathbf{N},\mathbf{U1}))} \arrow[d, "{(3,3),X}"] & & \\
 & & \boxed{((\mathbf{3},\mathbf{3}),(\mathbf{X},\emptyset,\emptyset,\emptyset,\mathbf{N},\mathbf{U2}))} \arrow[d, "{X,A}"] & & \\
 & & \boxed{((\mathbf{3},\mathbf{3}),(\mathbf{X},\emptyset,\bigcup_{\mathbf{W}\in\mathfrak{N}(\mathbf{A})}\mathbf{W},\emptyset,\mathbf{N},\mathbf{M}))} \arrow[dll, bend right, "{\begin{array}{c}(2,3),Q\\ (\neg c1\wedge\neg c2\text{ holds})\end{array}}"'] \arrow[dl, "{\begin{array}{c}(4,3),Q\\ (c1\text{ holds})\end{array}}"'] \arrow[dr, "{\begin{array}{c}(3,2),Q\\ (\neg c1\wedge c2\text{ holds})\end{array}}"] \arrow[drr, bend left, "{\begin{array}{c}(3,4),Q\\ (\neg c1\wedge c2\text{ holds})\end{array}}"] & & \\
\boxed{((\mathbf{2},\mathbf{3}),(\emptyset,\mathbf{X},\emptyset,\mathfrak{N}(\mathbf{Q}),\mathbf{N},\mathbf{U1}))} \arrow[d, "{(2,3),X'}"'] & \boxed{((\mathbf{4},\mathbf{3}),(\emptyset,\mathbf{X},\emptyset,\mathfrak{N}(\mathbf{Q}),\mathbf{S},\mathbf{U1}))} \arrow[d, "{(4,3),X'}"'] & & \boxed{((\mathbf{3},\mathbf{2}),(\emptyset,\mathbf{X},\emptyset,\mathfrak{N}(\mathbf{Q}),\mathbf{W},\mathbf{U1}))} \arrow[d, "{(3,2),X'}"] & \boxed{((\mathbf{3},\mathbf{4}),(\emptyset,\mathbf{X},\emptyset,\mathfrak{N}(\mathbf{Q}),\mathbf{E},\mathbf{U1}))} \arrow[d, "{(3,4),X'}"] \\
\boxed{(\mathbf{2},\mathbf{3}),(\mathbf{X}',\mathbf{X},\emptyset,\mathfrak{N}(\mathbf{Q}),\mathbf{N},\mathbf{U2}))} \arrow[d, dash, dotted] & \boxed{((\mathbf{4},\mathbf{3}),(\mathbf{X}',\mathbf{X},\emptyset,\mathfrak{N}(\mathbf{Q}),\mathbf{S},\mathbf{U2}))} \arrow[d, dash, dotted] & & \boxed{((\mathbf{3},\mathbf{2}),(\mathbf{X}',\mathbf{X},\emptyset,\mathfrak{N}(\mathbf{Q}),\mathbf{W},\mathbf{U2}))} \arrow[d, dash, dotted] & \boxed{((\mathbf{3},\mathbf{4}),(\mathbf{X}',\mathbf{X},\emptyset,\mathfrak{N}(\mathbf{Q}),\mathbf{E},\mathbf{U2}))} \arrow[d, dash, dotted] \\
 {} & {} & & {} & {}
\end{tikzcd}$$

Figure 4.4: A fragment of the agent template for the Beta swarm aggregation algorithm.

$$\begin{tikzcd}
\boxed{\{((\mathbf{3},\mathbf{3}),\emptyset,\emptyset,\emptyset,\emptyset,\mathbf{N},\mathbf{U1}),((\mathbf{3},\mathbf{3}),\emptyset,\emptyset,\emptyset,\emptyset,\mathbf{S},\mathbf{U1})\}} \arrow[d, "{\{((3,3),\{(3,3)\}),((3,3),\{(3,3)\})\}}"] \\
\boxed{\{((\mathbf{3},\mathbf{3}),\{(\mathbf{3},\mathbf{3})\},\emptyset,\emptyset,\emptyset,\mathbf{N},\mathbf{U2}),((\mathbf{3},\mathbf{3}),\{(\mathbf{3},\mathbf{3})\},\emptyset,\emptyset,\emptyset,\mathbf{S},\mathbf{U2})\}} \arrow[d, "{\{(\{(3,3)\},\{\{(3,3)\}\}),(\{(3,3)\},\{\{(3,3)\}\})\}}"] \\
\boxed{\{((\mathbf{3},\mathbf{3}),\{(\mathbf{3},\mathbf{3})\},\emptyset,\{(\mathbf{3},\mathbf{3})\},\emptyset,\mathbf{N},\mathbf{M}),((\mathbf{3},\mathbf{3}),\{(\mathbf{3},\mathbf{3})\},\emptyset,\{(\mathbf{3},\mathbf{3})\},\emptyset,\mathbf{S},\mathbf{M})\}} \arrow[d, "{\begin{array}{l}\{((2,3),\{(2,3),(4,3)\}),((4,3),\{(2,3),(4,3)\})\}\\ \neg c1 \land \neg c2 \text{ holds for both template states}\end{array}}"] \\
\boxed{\{((\mathbf{2},\mathbf{3}),\emptyset,\{(\mathbf{3},\mathbf{3})\},\emptyset,\{(\mathbf{2},\mathbf{3}),(\mathbf{4},\mathbf{3})\},\mathbf{N},\mathbf{U1}),((\mathbf{4},\mathbf{3}),\emptyset,\{(\mathbf{3},\mathbf{3})\},\emptyset,\{(\mathbf{2},\mathbf{3}),(\mathbf{4},\mathbf{3})\},\mathbf{S},\mathbf{U1})\}} \arrow[d, "{\{((2,3),\{(2,3)\}),((4,3),\{(4,3)\})\}}"] \\
\boxed{\{((\mathbf{2},\mathbf{3}),\{(\mathbf{2},\mathbf{3})\},\{(\mathbf{3},\mathbf{3})\},\emptyset,\{(\mathbf{2},\mathbf{3}),(\mathbf{4},\mathbf{3})\},\mathbf{N},\mathbf{U2}),((\mathbf{4},\mathbf{3}),\{(\mathbf{4},\mathbf{3})\},\{(\mathbf{3},\mathbf{3})\},\emptyset,\{(\mathbf{2},\mathbf{3}),(\mathbf{4},\mathbf{3})\},\mathbf{S},\mathbf{U2})\}} \arrow[d, "{\{(\{(2,3)\},\{\{(2,3)\}\}),(\{(4,3)\},\{\{(4,3)\}\})\}}"] \\
\boxed{\{((\mathbf{2},\mathbf{3}),\{(\mathbf{2},\mathbf{3})\},\{(\mathbf{3},\mathbf{3})\},\{(\mathbf{2},\mathbf{3})\},\{(\mathbf{2},\mathbf{3}),(\mathbf{4},\mathbf{3})\},\mathbf{N},\mathbf{M}),((\mathbf{4},\mathbf{3}),\{(\mathbf{4},\mathbf{3})\},\{(\mathbf{3},\mathbf{3})\},\{(\mathbf{4},\mathbf{3})\},\{(\mathbf{2},\mathbf{3}),(\mathbf{4},\mathbf{3})\},\mathbf{S},\mathbf{M})\}} \arrow[d, "{\begin{array}{l}\{((3,3),\{(3,3)\}),((3,3),\{(3,3)\})\}\\ c1 \text{ holds for both template states}\end{array}}"] \\
\boxed{\{((\mathbf{3},\mathbf{3}),\emptyset,\{(\mathbf{2},\mathbf{3})\},\emptyset,\{(\mathbf{3},\mathbf{3})\},\mathbf{S},\mathbf{U1}),((\mathbf{3},\mathbf{3}),\emptyset,\{(\mathbf{4},\mathbf{3})\},\emptyset,\{(\mathbf{3},\mathbf{3})\},\mathbf{N},\mathbf{U1})\}} \arrow[d, dotted, dash] \\
{}
\end{tikzcd}$$

Figure 4.5: A fragment of the abstract model for the Beta swarm aggregation algorithm.

- For $(x', y') \in Loc$ and $p = M$, we have $(x', y') \in P(s)$ iff either one of the following holds:
  - $c1$ holds on s (there is at least one neighbour that has in its neighbourhood a lost robot) and (x', y') is one cell to the opposite direction of d from (x, y);
  - $\neg c1 \wedge c2$ holds on s (the number of neighbouring occupied cells is greater than said number in the previous step) and (x', y') is one cell to direction d' from cell (x, y), where d' is either 90° to the left or $90°$ to the right of d.
  - $\neg c1 \wedge \neg c2$ holds on s and (x', y') is one cell to direction d from (x, y).

We now define the transition function $t : L \times Act \times \mathcal{P}(Act) \to L$. Let $s = ((x, y), (X, Y, Z, Q, d, p))$ and $s' = ((x', y'), (X', Y', Z', Q', d', p'))$ be two states. For a set A of actions, let $\mathfrak{N}(A)$ denote the set of actions in A performed by agents occupying cells in the neighbourhood of (x, y); intuitively, the transition function of a robot depends only on the actions performed by robots in its neighbourhood. We have

$$t(((x, y), (X, Y, Z, Q, d, p)), a, A) = ((x', y'), (X', Y', Z', Q', d', p'))$$

iff one of the following holds:

- $p = U1$, $(x', y') = (x, y), X' = \mathfrak{N}(A), Y' = Y, Z' = Z, Q' = Q, d' = d$, and $p' = U2$;
- $p = U2$, $(x', y') = (x, y), X' = X, Y' = Y, Z' = \bigcup_{W \in \mathfrak{N}(A)} W, Q' = Q, d' = d$, and $p' = M$;
- $p = M$, $(x', y') = a$, $X' = \emptyset$, $Y' = X$, $Z' = \emptyset$, $Q' = \mathfrak{N}(A)$, d' is the direction from (x, y) to a, and $p' = U1$.

Finally, we assume a unique initial position $(3, 3)$ in the arena and a set

$$I = \{((3, 3), (\emptyset, \emptyset, \emptyset, \emptyset, d, U1)) \colon d \in \{North, East, South, West\}\}$$

of initial states; thus, a robot has no initial information regarding its neighbours.

Having encoded the Beta algorithm as a PIS, we can now express properties in indexed ACTLK. We say that a robot i is connected to a robot j if j is in the transitive closure of i's neighbours.

Note that, given the size of the arena and the communication range, i and j are connected in a global state iff they agree on the sets Z of cells encoded in their local state. We associate a fresh atomic proposition w for each set W of cells, and we assign w to each template state with $W = Z$. We consider the formula

$$\phi = \forall_{\{v,u\}} K^{v}_{R_\beta} AFAG \bigvee_{W} ((w, v) \wedge (w, u))$$

expressing "every robot knows that eventually it will be forever connected to every other robot". This means that not only will the whole swarm be connected in a single cluster, but everyone will know to be connected. We are interested to check whether ϕ emerges in $\mathcal{S}_\beta$ as the number of robots increases in the system.

Following the PIS procedure, we can compute a cutoff of 16. This is done as follows. We first observe that ϕ is a 2-indexed formula. Therefore, in step 1, we build the abstract model $\hat{\mathcal{S}}_\beta(2)$. A fragment of $\hat{\mathcal{S}}_\beta(2)$ for the initial state in which the robots' direction of movement is to the north and south is depicted in Figure 4.5. For ease of presentation, in the figure the concrete component of each abstract state is omitted. For each abstract transition $(\{s, s'\}, \{(a, A), (a', A')\}, \{s^1, s'^1\})$, the transition labels (a, A) and (a', A') are a shortcut for (s, a, A, s^1) and (s', a', A', s'^1), respectively. In step 2, we build the pruned computation forest of $\hat{\mathcal{S}}_\beta(2)$ to not include repetitions of a cycle for more than 3 times. Finally, in step 3, we search for a concrete system that simulates the forest. It can be established that the concrete system of 16 robots is as required. Thus, we conclude that ϕ is satisfied in $\mathcal{S}_\beta$ iff $\mathcal{S}_\beta(2) \models \phi, \ldots, \mathcal{S}_\beta(16) \models \phi$. The latter queries can be put to any epistemic model checker, which would return false, thereby establishing that ϕ is not satisfied in $\mathcal{S}_\beta$, but also that ϕ is not satisfied by any concrete system with more than 16 robots. It follows that the robots will not eventually congregate into a stable cluster.

We conclude that the property above is not satisfied by the Beta protocol. We emphasise that the above result cannot be obtained with traditional model checking nor simulation techniques since an unbounded number of systems would have to be considered.

Chapter 5

Verifying parameterised interleaved interpreted systems

This chapter develops a parameterised model checking procedure for the verification of unbounded multiagent systems represented in the formalism of parameterised interleaved interpreted systems. Section 5.1 gives a formal definition of the parameterised model checking problem and of the notion of cutoffs for parameterised interleaved interpreted systems. Section 5.2 introduces the `PIIS` procedure. The procedure identifies a cutoff for a given system and a given specification. A cutoff expresses the number of agents that is sufficient to consider when evaluating a given specification. Following the cutoff identification, the procedure solves the parameterised model checking problem by checking all concrete systems up to the cutoff. Finally, section **??** applies the procedure to the Train-Gate-Controller model and the Alpha swarm aggregation algorithm.

5.1 Parameterised model checking problem

We outline parameterised interleaved interpreted systems as presented in Section 3.2. We then define the parameterised model checking problem for PIIS. Finally, we define the notion of cutoffs in the context of PIIS.

A PIIS is a tuple

$$PIIS = (\mathcal{T}, \mathcal{E}, \mathcal{V}),$$

where $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\} = \{(L_1, \iota_1, Act_1, P_1, t_1), \ldots, (L_k, \iota_k, Act_k, P_k, t_k)\}$ is a set of $k \geq 1$ agent templates and $\mathcal{E} = (L_\mathcal{E}, \iota_\mathcal{E}, Act_\mathcal{E}, P_\mathcal{E}, t_\mathcal{E})$ is an environment template. An agent template is associated with a set L_i of template states, an initial template state $\iota_i \in L_i$, a set Act_i of template actions, a template protocol $P_i : L_i \to \mathcal{P}(Act_i)$, and a template transition function $t_i : L_i \times Act_i \to L_i$. Each template $\mathcal{T}_i$ distinguishes between five types of actions: (i) asynchronous A_i actions; (ii) agent-environment AE_i actions; (iii) role-synchronous RS_i actions; (iv) global-synchronous GS actions; (v) multi-role MR_i actions. Specifically, the concrete agents from template $\mathcal{T}_i$ may evolve asynchronously via A_i actions, communicate with the environment via AE_i actions, synchronise with the agents of the same role via RS_i actions, synchronise with all the agents in the system via GS actions, and communicate with an agent performing a different role via MR_i actions. The environment template is similarly associated with a set $L_\mathcal{E}$ of template states, an initial template state $\iota_\mathcal{E} \in L_\mathcal{E}$, the set $Act_\mathcal{E} = \bigcup_{\mathcal{T}_i}(AE_i \cup RS_i \cup MR_i) \cup GS$ of template actions, a template protocol $P_\mathcal{E} : L_\mathcal{E} \to \mathcal{P}(Act_\mathcal{E})$, and a template transition function $t_\mathcal{E} : L_\mathcal{E} \times Act_\mathcal{E} \to L_\mathcal{E}$. The definition of a PIIS concludes with the description of a set of valuation functions on the template states $\mathcal{V} = \{V_i : L_i \to \mathcal{P}(AP_i) \colon 1 \leq i \leq k\}$.

Given a value $\overline{n} = (n_1, \ldots, n_k)$ of the system's parameter, the concrete system $PIIS(\overline{n})$ is the interleaved interpreted system

$$PIIS(\overline{n}) = \Big((L_i^j, \iota_i^j, Act_i^j, P_i^j, t_i^j)_{(i,j)\mathcal{A}(\overline{n})}, (L_E, \iota_E, Act_E, P_E, t_E), \iota, V \Big).$$

$PIIS(\overline{n})$ results from the composition of $\overline{n}.i$ instantiations $\{(i,1), \ldots, (i, \overline{n}.i)\}$ of each agent template $\mathcal{T}_i$ and an instantiation of the environment template. Given $\overline{x} \leq \overline{n}$, we write $\mathcal{A}(\overline{x})$ for the set $\mathcal{A}(\overline{x}) = \{(i,j) \colon 1 \leq i \leq k, 1 \leq j \leq \overline{x}.i\}$ of concrete agents. For each concrete system $PIIS(\overline{n})$ generated from the templates we can associate a temporal-epistemic model $\mathcal{S}_{PIIS(\overline{n})} = \Big(G(\overline{n}), \iota(\overline{n}), R(\overline{n}), (\mathcal{K}_i^j)_{(i,j) \in \mathcal{A}(\overline{n})}, V(\overline{n}) \Big)$ as standard (see Section 2.1.1). When $PIIS(\overline{n})$ is clear from the context we simply write $\mathcal{S}(\overline{n})$ for $\mathcal{S}_{PIIS(\overline{n})}$. We assume that the joint null action is always enabled in a concrete system. Therefore, the concrete transition relations are serial.

We now proceed to state the parameterised model checking problem for PIIS.

Definition 5.1 (Parameterised model checking problem for PIIS). *Given a PIIS $\mathcal{S}$ and an $\overline{m}$-indexed ACTL*K$\backslash X$ formula $\forall_{\overline{v}}\phi(\overline{v})$, the parameterised model checking problem concerns establishing whether or not the following holds:*

$$\forall \overline{n} \geq \overline{m}.\, \mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v})$$

If the above holds, then $\forall_{\overline{v}}\phi(\overline{v})$ is said to be satisfied by $\mathcal{S}$. This is denoted by $\mathcal{S} \models \forall_{\overline{v}}\phi(\overline{v})$.

A cutoff for a given PIIS $\mathcal{S}$ and an $\overline{m}$-indexed formula $\forall_{\overline{v}}\phi(\overline{v})$ is the number of components that is sufficient to consider when solving the parameterised model checking problem for $\mathcal{S}$ and $\forall_{\overline{v}}\phi(\overline{v})$.

Definition 5.2 (MAS cutoff). *Given a PIIS $\mathcal{S}$ composed of k roles and an $\overline{m}$-indexed ACTL*K$\backslash X$ formula $\forall_{\overline{v}}\phi(\overline{v})$, a k-tuple $\overline{c} \in \mathbb{N}^k$ is said to be a* MAS cutoff *if the following holds:*

$$\forall \overline{m} \leq \overline{x} \leq \overline{c}.\, \mathcal{S}(\overline{c}) \models \forall_{\overline{v}}\phi(\overline{v}) \textit{ if and only if } \forall \overline{n} \geq \overline{c}.\, \mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v})$$

We say that $\mathcal{S}$ admits a cutoff for $\forall_{\overline{v}}\phi(\overline{v})$ and we call $\mathcal{S}(\overline{c})$ the cutoff system.

The aim of this chapter is to develop a sound procedure that takes as input a PIIS $\mathcal{S}$ and an $\overline{m}$-indexed formula $\forall_{\overline{v}}\phi(\overline{v})$, and returns a cutoff for $\mathcal{S}$ and $\forall_{\overline{v}}\phi(\overline{v})$.

5.2 The `PIIS` procedure

This section introduces the `PIIS` procedure. The procedure tackles the verification of parameterised interleaved interpreted systems against indexed ACTL*K$\backslash X$ formulae.

5.2.1 Overview

Given a PIIS $\mathcal{S}$ and an $\overline{m}$-indexed ACTL*K$\backslash X$ formula $\forall_{\overline{v}}\phi(\overline{v})$, the procedure identifies a cutoff $\overline{c}$ for $\mathcal{S}$ and $\forall_{\overline{v}}\phi(\overline{v})$. Following this, the procedure checks every concrete system up to $\mathcal{S}(\overline{c})$ against $\forall_{\overline{v}}\phi(\overline{v})$. If the formula is satisfied on each of the systems checked, then the procedure concludes that the formula is satisfied on every concrete system. The cutoff identification is performed by means of the following two steps.

1. In step 1, `PIIS` builds an abstract interleaved interpreted system. The abstraction generates abstract states that reflect the set of template states that arbitrarily many participants may be in in a concrete global state. Similarly to the abstract model introduced for PIS, the abstract system can simulate any concrete system from $\mathcal{S}$ of arbitrary size. Following this, the satisfiability of $\forall_{\overline{v}}\phi(\overline{v})$ on the abstract system implies the satisfiability of $\forall_{\overline{v}}\phi(\overline{v})$ on every concrete system. However, the abstract system may contain spurious paths. Therefore, if $\forall_{\overline{v}}\phi(\overline{v})$ is not satisfied by the abstract system, then it does not necessarily follow that there is a concrete system not satisfying $\forall_{\overline{v}}\phi(\overline{v})$. Differently from the abstract model for PIS, however, spurious paths in the abstract interleaved interpreted system cannot be characterised. In fact, the parameterised model checking problem for PIIS is undecidable. This is because PIIS can simulate broadcast protocols for which the parameterised model checking problem for liveness properties was shown to be undecidable [EFM99]. Thus, the procedure checks whether the abstract system satisfies $\forall_{\overline{v}}\phi(\overline{v})$. If so, it then terminates. Otherwise, it realises step 2.

2. In step 2, `PIIS` iteratively checks whether there is an $\overline{x} \geq \overline{m}$ such that the concrete system $\mathcal{S}(\overline{x})$ *gs-simulates* the abstract system. Roughly speaking, the existence of a gs-simulation between the abstract system and a concrete system suggests that every agent in every concrete system can always succeed in globally synchronising. In other words, if an agent can perform a global-synchronous or a role-synchronous action at a round, then every other agent can eventually move to a state in which the action can be performed. If this property is admitted by $\mathcal{S}$, then said concrete system exists and is the cutoff system.

In the following sections we give a detailed description of the above steps.

5.2.2 Step 1: abstraction

The abstract model corresponds to the application of the abstraction ideas discussed in Section 4.2 to PIISs. Given a PIIS $\mathcal{S}$ and an $\overline{m}$-indexed formula $\forall_{\overline{v}}\phi(\overline{v})$, the abstract model is an interleaved interpreted system composed of $\mathcal{A}(\overline{m})$ concrete agents of $\mathcal{S}$, and an abstract agent for each agent template. An abstract state is made of a concrete component and an abstract component. The concrete component consists of the local states of the agents in $\mathcal{A}(\overline{m})$, whereas the abstract component consists of the local states of the abstract agents. A local state of the abstract agent for template $\mathcal{T}_i$ consists of a set of template local states that is the projection of the tuple of local states of all the concrete agents from $\mathcal{T}_i$ other than the ones in $\mathcal{A}(\overline{m})$ into a set. Actions of an abstract agent enabled at an abstract state represent concrete actions enabled at any concrete state represented by said abstract state.

We first define the abstract agents. Then, we define the abstract system.

Definition 5.3 (Abstract agent)**.** *Given a PIIS $\mathcal{S}$ of k roles and $\overline{m} \in \mathbb{N}^k$, the* abstract agent $\hat{a}_i = \left(\hat{L}_i, \hat{\iota}_i, \hat{Act}_i, \hat{P}_i, \hat{t}_i\right)$ *of agent template $\mathcal{T}_i$ is defined as follows.*

- $\hat{L}_i = \mathcal{P}(L_i) \setminus \emptyset$ *is the set of abstract local states.*
- $\hat{\iota}_i = \{\iota_i\}$ *is the initial abstract local state.*
- $\hat{Act}_i = \hat{A}_i \cup \hat{AE}_i \cup \hat{RS}_i \cup \hat{GS} \cup \hat{MR}_i \cup \{\hat{\epsilon}_i\}$ *is the set of abstract actions, where:*
  - $\hat{A}_i = A_i \times (L_i \times \{\downarrow, \uparrow\})$;
  - $\hat{AE}_i = AE_i \times (L_i \times \{\downarrow, \uparrow\})$;
  - $\hat{RS}_i = RS_i$;
  - $\hat{GS} = GS$;
  - $\hat{MR}_i = \bigcup_{1 \le j \le k} \hat{MR}^{concrete}_{i,j} \cup \hat{MR}^{abstract}_{i,j}$, *where*
    - $\hat{MR}^{concrete}_{i,j} = MR_{i,j} \times (L_i \times \{\downarrow, \uparrow\}) \times \{(j, 1), \ldots, (j, \overline{m}.j)\}$;
    - $\hat{MR}^{abstract}_{i,j} = MR_{i,j} \times \{\{(l, x), (l', y)\} : l \in L_i, l' \in L_j, x, y \in \{\downarrow, \uparrow\}\}$;
  - $\hat{\epsilon}_i$ *is the abstract null action.*
- $\hat{P}_i : \hat{L}_i \to \mathcal{P}(\hat{Act}_i)$ *is the abstract protocol defined as*

$$P_i(\hat{l}) = \left\{a \colon a_{\mathcal{T}} \in P_i(l) \text{ for some } l \in \hat{l}\right\},$$

where $a_{\mathcal{T}}$ denotes the corresponding template action from which a was obtained.

- *$\hat{t}_i : \hat{L}_i \times \hat{Act}_i \to \hat{L}_i$ is the abstract evolution function defined as follows: $\hat{t}_i(\hat{l}, a) = \hat{l}'$ iff $a \in \hat{P}_i(\hat{l})$ and one of the following holds:*

  1. *if a is an asynchronous action, an agent-environment action, or a multi-role action that is indexed by $(l, \downarrow) \in L_i \times \{\downarrow\}$, then $\hat{l}' = (\hat{l} \setminus \{l\}) \cup \{t_i(l, a)\}$;*
  2. *if x is an asynchronous action, an agent-environment action, or a multi-role action that is indexed by $(l, \uparrow) \in L_i \times \{\uparrow\}$, then $\hat{l}' = \hat{l} \cup \{t_i(l, a)\}$;*
  3. *if a is a role-synchronous action or a global-synchronous action, then $\hat{l}' = \{l' \in L_i : \exists l \in \hat{l}.t_i(l, a) = l'\}$.*

A local state $\hat{l}$ of an abstract agent $\hat{a}_i$ represents any global state g in any concrete system $\mathcal{S}(\overline{n})$ (with $\overline{n} \geq \overline{m}$) in which $\hat{l} = \left\{tls_i^j(g) \colon j \in \{\overline{m}.i + 1, \ldots, \overline{n}.i\}\right\}$ is the projection of the tuple of the template local states for the agents $(i, \overline{m}.i + 1), \ldots, (i, \overline{n}.i)$ into a set.

The actions of an abstract agent are made of the template actions of the corresponding agent template. They represent all of their concrete instantiations that the agents $\overline{m}.i + 1, \ldots, \overline{n}.i$ may perform at a global state.

Assume a concrete asynchronous action or a concrete agent-environment action (a, j) of agent (i, j). To respect the encoding of the abstract states, two cases have to be considered when representing a global transition by means of a: (i) there is exactly one concrete agent from $\mathcal{T}_i$ in template state l at which a is performed; (ii) there are at least two agents. The former case is represented by the action $(a, l, \downarrow)$, whereas the latter is represented by the action $(a, l, \uparrow)$. The former action causes the abstract state to "shrink" ($\downarrow$) and not include l, whereas the latter action causes the abstract state to "grow" ($\uparrow$) and include the template state l' as specified by the template transition $t_i(l, a) = l'$.

Now assume a concrete multi-role action $(a, \{(i, j), (r, q)\})$ shared between the agents (i, j) and (r, q). Suppose that (i, j) is at template state l and (r, q) is at template state l'. To respect the encoding of the abstract states, 6 cases have to be considered when representing a global transition by means of a. The first two cases occur whenever one of the concrete agents, say agent (i, j), is an agent in $\mathcal{A}(\overline{m})$ and the other is not in $\mathcal{A}(\overline{m})$. For the abstract system to simulate the global transition, the concrete agent (i, j) and the abstract agent $\hat{a}_r$

have to update their states accordingly. To do this, they synchronise on the $MR_{r,i}^{\hat{concrete}}$ action $(a, (l', \downarrow), (i, j))$ or the $MR_{r,i}^{\hat{concrete}}$ action $(a, (l', \uparrow), (i, j))$, depending on whether the state of abstract agent r should shrink or grow; as we show below, these actions are shared precisely between the agent (i, j), the abstract agent $\hat{a}_r$, and the environment. The other four cases occur whenever both (i, j) and (r, q) are not in $\mathcal{A}(\overline{m})$. For the abstract system to simulate the global transition, the abstract agent $\hat{a}_i$ and the abstract agent $\hat{a}_r$ have to update their states accordingly. To do this, they synchronise on either one of the $MR_{i,r}^{\hat{abstract}} = MR_{r,i}^{\hat{abstract}}$ actions $(a, \{(l, \downarrow), (l', \downarrow)\})$, $(a, \{(l, \downarrow), (l', \uparrow)\})$, $(a, \{(l, \uparrow), (l', \downarrow)\})$, $(a, \{(l, \uparrow), (l', \uparrow)\})$, depending on whether their states should shrink or grow.

We now define the abstract system. The system is composed of $\mathcal{A}(\overline{m})$ concrete agents of $\mathcal{S}$, and an abstract agent for each agent template. For ease of the definition, we first define the extension of the concrete system $\mathcal{S}(\overline{m})$ to include the abstract actions mentioned above. These actions enable the synchronisation between the concrete agents in $\mathcal{A}(\overline{m})$, the concrete environment, and the abstract agents. In particular, the extended system extends the agents in $\mathcal{S}(\overline{m})$ to include the abstract multi-role actions. Additionally, the system extends the environment to include the abstract agent-environment actions and the abstract multi-role actions.

Intuitively, as mentioned above, an abstract agent $\hat{a}.i$ represents all possible synchronisation patterns involving the agents $(i, \overline{m}.i + 1), \ldots, (i, \overline{n}.i)$. These agents may communicate with an agent $(r, q) \in \mathcal{A}(\overline{m})$ via a multi-role action. By extending the agent (r, q) with the abstract actions, this can be represented in the abstract system with the abstract agent and the agent (r, q) performing the corresponding abstract action. Similarly, whenever one of the agents $(i, \overline{m}.i + 1), \ldots, (i, \overline{n}.i)$ performs an agent-environment action or a multi-role action, the environment participates in the global transition. By extending the environment with the abstract actions, the participation of the environment in the global transition can be represented in the abstract system. The abstract system can then be defined as the composition of the extended system and the abstract agents.

Definition 5.4 (Extended concrete system). *Given a PIIS $\mathcal{S}$ and $\overline{m} \in \mathbb{N}^k$, consider the concrete system*

$$\mathcal{S}(\overline{m}) = \Big((L_i^j, \iota_i^j, Act_i^j, P_i^j, t_i^j)_{(i,j)\mathcal{A}(\overline{m})}, (L_E, \iota_E, Act_E, P_E, t_E), \iota, V \Big) .$$

The extended concrete system $\tilde{\mathcal{S}}(\overline{m})$ is the interleaved interpreted system

$$\tilde{\mathcal{S}}(\overline{m}) = \Big((L_i'^j, \iota_i'^j, Act_i'^j, P_i'^j, t_i'^j)_{(i,j)\mathcal{A}(\overline{m})}, (L_E', \iota_E', Act_E', P_E', t_E'), \iota, V \Big) ,$$

where each agent $(i, j) = (L_i'^j, \iota_i'^j, Act_i'^j, P_i'^j, t_i'^j)$ is defined as follows:

- $L_i'^j = L_i^j$;
- $\iota_i'^j = \iota_i^j$;
- $Act_i'^j$ *extends* Act_i^j *by including all abstract actions that are indexed with agent* (i, j);
- *The protocol* $P_i'^j$ *is as* P_i^j, *but defined on the abstract actions as follows:* $a \in P_i'^j(l)$ *iff* $a_\mathcal{T} \in P_i(l)$ *(where* $a_\mathcal{T}$ *is the template action from which* a *was obtained);*
- *The evolution function* $t_i'^j$ *is as* t_i^j, *but defined on the abstract actions as follows:* $t_i'^j(l, a) = l'$ *iff* $t_i(l, a_\mathcal{T}) = l'$.

The environment $E' = (L_E', \iota_E', Act_E', P_E', t_E')$ is given by the following:

- $L_E' = L_E$;
- $\iota_E' = \iota_E$;
- Act_E' *extends* Act_E *by including all agent-environment and all multi-role actions of the every abstract agent;*
- *The abstract protocol* P_E' *is as* P_E, *but defined on the abstract actions as follows:* $a \in P_E'(l)$ *iff* $a_\mathcal{T} \in P_\mathcal{E}(l)$;
- *The abstract evolution function* t_E' *is as* t_E, *but defined on the abstract actions as follows:* $t_E'(l_E, a) = l_E'$ *iff* $t_\mathcal{E}(l_E, a_\mathcal{T}) = l_E'$;

We now have all the necessary ingredients to define the abstract system.

Definition 5.5 (Abstract interleaved interpreted system). *Given a PIIS $\mathcal{S}$ of k roles and $\overline{m} \in \mathbb{N}^k$, the abstract system is the interleaved interpreted system*

$$\hat{\mathcal{S}}(\overline{m}) = \Big(\hat{G}(\overline{m}), \hat{\iota}(\overline{m}), \hat{R}(\overline{m}), (\hat{\mathcal{K}}_i^j)_{(i,j)\in\mathcal{A}(\overline{m})}, \hat{V}(\overline{m}) \Big)$$

composed of the agents in the extended system $\tilde{\mathcal{S}}(\overline{m})$ and the abstract agents $\hat{a}_1, \ldots, \hat{a}_k$. The abstract valuation function $\hat{V}(m)$ is given by

$$p \in \hat{V}(\overline{m})\left(\left(l_1^1, \ldots, l_k^{\overline{m}.k}, \hat{l}_1, \ldots, \hat{l}_k\right)\right) \text{ iff } p \in V(\overline{m})\left(\left(l_1^1, \ldots, l_k^{\overline{m}.k}\right)\right),$$

where $V(\overline{m})$ is the valuation function for $\mathcal{S}(\overline{m})$.

Given an abstract state γ, we write $ls_i^j(\gamma)$ for the local state of agent (i,j) in γ. We denote the local state of abstract agent $\hat{a}_i$ in γ by $ls_{\hat{i}}(\gamma)$.

Example 5.1. *In Section 3.2.1 we represented the Train-Gate-Controller in the formalism of parameterised interleaved interpreted systems. We encoded the scenario by means of an agent template representing prioritised trains, an agent template representing normal trains, and the environment representing the controller. A fragment of the concrete system with two prioritised trains and two normal trains was given in Figure 3.7. Figure 5.1 gives a fragment of the abstract system for $(1,1)$-indexed formulae. In the figure an abstract state is of the form (l, X, e, l', X'), where l is the local state of the concrete prioritised train, X is the local state of the abstract prioritised agent, e is the local state of the environment, l' is the local state of the concrete normal train, and X' is the local state abstract normal train; W stands for $WAIT$, PG for P_GREEN, L for $TUNNEL\_LOCKED$, T for $TUNNEL$, R for RED, A for $AWAY$, and NG for N_GREEN.*

Correspondence between a concrete system and the abstract system

Assume a PIIS $\mathcal{S}$ of k roles and $\overline{m} \in \mathbb{N}^k$. We now establish a correspondence between a concrete system $\mathcal{S}(\overline{n})$ with $\overline{n} \geq \overline{m}$ and $\hat{\mathcal{S}}(\overline{m})$. Define $\delta_{\overline{n}} : G(\overline{n}) \to \hat{G}(\overline{m})$ to map concrete states in $\mathcal{S}(\overline{n})$ to abstract states in $\hat{\mathcal{S}}(\overline{m})$:

$$\delta_{\overline{n}}(g) = \left(ls_1^1(g), \ldots, ls_k^{\overline{m}.k}(g), ls_E(g), \hat{l}_1, \ldots, \hat{l}_k\right),$$

where

$$\hat{l}_1 = \left\{tls_1^j(g) \colon j \in \{\overline{m}.1 + 1, \ldots, \overline{n}.1\}\right\}, \ldots, \hat{l}_k = \left\{tls_k^j(g) \colon j \in \left\{\overline{k}.1 + 1, \ldots, \overline{n}.k\right\}\right\}.$$

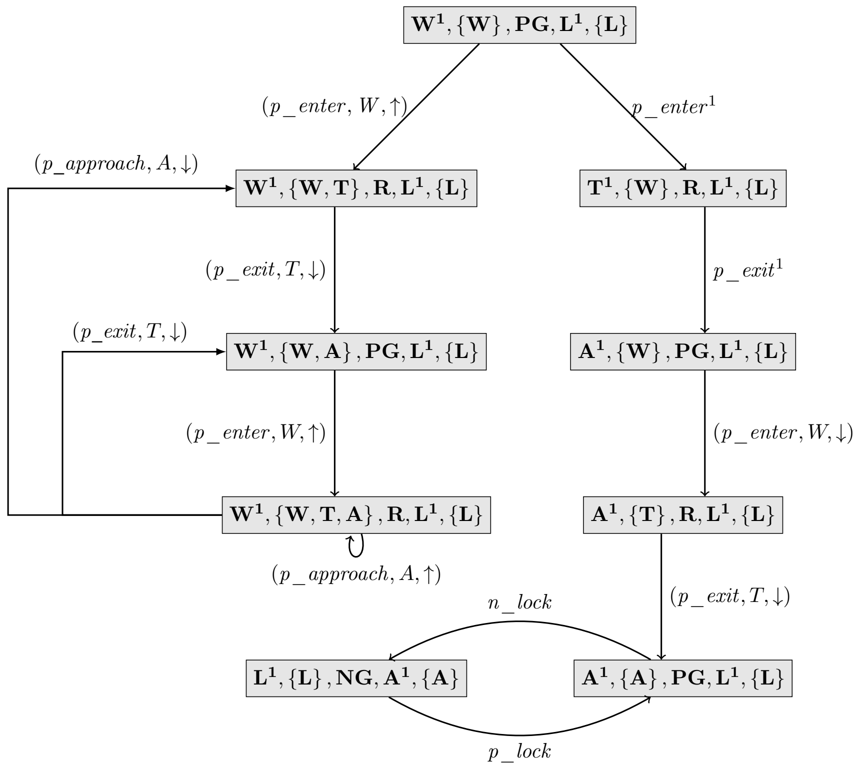

$\mathbf{W^1}, \{\mathbf{W}\}, \mathbf{PG}, \mathbf{L^1}, \{\mathbf{L}\}$
$(p\_enter, W, \uparrow)$
p_enter^1
$(p\_approach, A, \downarrow)$
$\mathbf{W^1}, \{\mathbf{W}, \mathbf{T}\}, \mathbf{R}, \mathbf{L^1}, \{\mathbf{L}\}$
$\mathbf{T^1}, \{\mathbf{W}\}, \mathbf{R}, \mathbf{L^1}, \{\mathbf{L}\}$
$(p\_exit, T, \downarrow)$
p_exit^1
$(p\_exit, T, \downarrow)$
$\mathbf{W^1}, \{\mathbf{W}, \mathbf{A}\}, \mathbf{PG}, \mathbf{L^1}, \{\mathbf{L}\}$
$\mathbf{A^1}, \{\mathbf{W}\}, \mathbf{PG}, \mathbf{L^1}, \{\mathbf{L}\}$
$(p\_enter, W, \uparrow)$
$(p\_enter, W, \downarrow)$
$\mathbf{W^1}, \{\mathbf{W}, \mathbf{T}, \mathbf{A}\}, \mathbf{R}, \mathbf{L^1}, \{\mathbf{L}\}$
$\mathbf{A^1}, \{\mathbf{T}\}, \mathbf{R}, \mathbf{L^1}, \{\mathbf{L}\}$
$(p\_approach, A, \uparrow)$
n_lock
$(p\_exit, T, \downarrow)$
$\mathbf{L^1}, \{\mathbf{L}\}, \mathbf{NG}, \mathbf{A^1}, \{\mathbf{A}\}$
$\mathbf{A^1}, \{\mathbf{A}\}, \mathbf{PG}, \mathbf{L^1}, \{\mathbf{L}\}$
p_lock

Figure 5.1: Fragment of the abstract system for the parameterised interleaved interpreted system of the Train-Gate-Controller.

Assume a concrete transition (g, a, g') from state g to state g' by means of action a. Consider $\zeta_{\overline{n}}$ to map (g, a, g') to an action in $\hat{\mathcal{S}}(\overline{m})$ as follows.

- if $a = (x, j)$ is an asynchronous action or an agent-environment action performed at template state l by the agent (i, j), then:
  - if $(i, j) \in \mathcal{A}(\overline{m})$, then $\zeta_{\overline{n}}(g, a, g') = a$;
  - otherwise, if $(i, j) \notin \mathcal{A}(\overline{m})$, then:
    * if none of the agents $(i, \overline{m}.i + 1), \ldots, (i, \overline{n}.i)$ is at template state l in g', then $\zeta_{\overline{n}}(a) = (x, l, \downarrow)$.
    * if at least one of the agents $(i, \overline{m}.i + 1), \ldots, (i, \overline{n}.i)$ is at template state l in g', then $\zeta_{\overline{n}}(a) = (x, l, \uparrow)$.
- if a is a role-synchronous action or a global-synchronous action, then $\zeta_{\overline{n}}(g, a, g') = a$.
- if $a = (x, \{(i, j), (r, q)\})$ is a multi-role action performed at template state l by the agent (i, j) and at template state l' by the agent (r, q), then:

- if $(i,j) \in \mathcal{A}(\overline{m})$, $(r,q) \in \mathcal{A}(\overline{m})$, then $\zeta_{\overline{n}}(g,a,g') = a$.
- if $(i,j) \in \mathcal{A}(\overline{m})$, $(r,q) \notin \mathcal{A}(\overline{m})$, then:
  - if none of the agents $(r, \overline{m}.r+1), \ldots, (r, \overline{n}.r)$ is at template state l' in g', then $\zeta_{\overline{n}}(g,a,g') = (a, (l', \downarrow), (i,j))$.
  - if at least one of the agents $(r, \overline{m}.r+1), \ldots, (r, \overline{n}.r)$ is at template state l' in g', then: $\zeta_{\overline{n}}(g,a,g') = (a, (l', \uparrow), (i,j))$.
- the case where $(i,j) \notin \mathcal{A}(\overline{m})$, $(r,q) \in \mathcal{A}(\overline{m})$ is similar to the above case.
- if $(i,j) \notin \mathcal{A}(\overline{m})$, $(r,q) \notin \mathcal{A}(\overline{m})$, then:
  - if none of the agents $(i, \overline{m}.i+1), \ldots, (i, \overline{n}.i)$ is at template state l in g', and none of the agents $(r, \overline{m}.r+1), \ldots, (r, \overline{n}.r)$ is at template state l' in g', then $\zeta_{\overline{n}}(g,a,g') = (a, \{(l, \downarrow), (l', \downarrow)\})$;
  - if none of the agents $(i, \overline{m}.i+1), \ldots, (i, \overline{n}.i)$ is at template state l in g', and at least one of the agents $(r, \overline{m}.r+1), \ldots, (r, \overline{n}.r)$ is at template state l' in g', then $\zeta_{\overline{n}}(g,a,g') = (a, \{(l, \downarrow), (l', \uparrow)\})$.
  - if at least one of the agents $(i, \overline{m}.i+1), \ldots, (i, \overline{n}.i)$ is at template state l in g', and none of the agents $(r, \overline{m}.r+1), \ldots, (r, \overline{n}.r)$ is at template state l' in g', then $\zeta_{\overline{n}}(g,a,g') = (a, \{(l, \uparrow), (l', \downarrow)\})$;
  - if at least one of the agents $(i, \overline{m}.i+1), \ldots, (i, \overline{n}.i)$ is at template state l in g', and at least one of the agents $(r, \overline{m}.r+1), \ldots, (r, \overline{n}.r)$ is at template state l' in g', then $\zeta_{\overline{n}}(g,a,g') = (a, \{(l, \uparrow), (l', \uparrow)\})$.

By means of the above mappings $\hat{\mathcal{S}}(\overline{m})$ can simulate $\mathcal{S}(\overline{n})$, as the following shows.

Lemma 5.1. *Let $\mathcal{S}(\overline{n})$ be a concrete system with $\overline{n} \geq \overline{m}$. Then, $\mathcal{S}(\overline{n}) \leq_s \hat{\mathcal{S}}(\overline{m})$.*

Proof. Define the relation $\sim_s = \{(g,\gamma) \colon \delta_{\overline{n}}(g) = \gamma\}$ between $\mathcal{S}(\overline{n})$ and $\hat{\mathcal{S}}(\overline{m})$. We show that $\sim_s$ is a simulation relation between $\mathcal{S}(\overline{n})$ and $\hat{\mathcal{S}}(\overline{m})$. It should be clear that $(\iota(\overline{n}), \hat{\iota}(\overline{m})) \in \sim_s$, where $\iota(\overline{n}), \hat{\iota}(\overline{m})$ are the initial states of $\mathcal{S}(\overline{n})$ and $\hat{\mathcal{S}}(\overline{m})$, respectively. Let $(g,\gamma) \in \sim_s$ be an arbitrary pair of states. We have to show the following:

(i) $V(\overline{n})(g) = \hat{V}(\overline{m})(\gamma)$.

The requirement follows from $\delta_{\overline{n}}(g) = \gamma$.

(ii) if $\mathcal{K}_i^j(g, g')$, for some agent $(i, j) \in \mathcal{A}(\overline{m})$, then $\hat{\mathcal{K}}_i^j(\gamma, \gamma')$ for some γ' with $(g', \gamma') \in \sim_s$. Assume that $\mathcal{K}_i^j(g, g')$, for some $(i, j) \in \mathcal{A}(\overline{m})$. Assume a path $g^1 a^1 \ldots a^{x-1} g^x$ with $g^1 = \iota(\overline{n})$ and $g^x = g'$. Consider the path $\delta_{\overline{n}}(g^1)\zeta_{\overline{n}}(g^1, a^1, g^2) \ldots \zeta_{\overline{n}}(g^{x-1}, a^{x-1}, g^x)\delta_{\overline{n}}(g^x)$ in $\hat{\mathcal{S}}(\overline{m})$ obtained by performing the sequence of actions $\zeta_{\overline{n}}(g^1, a^1, g^2) \ldots \zeta_{\overline{n}}(g^{x-1}, a^{x-1}, g^x)$. It should be clear that this is a valid path in $\hat{\mathcal{S}}(\overline{m})$. We have that $\hat{\mathcal{K}}_i^j(\gamma, \delta_{\overline{n}}(g^x))$ and $(g', \delta_{\overline{n}}(g^x)) \in \sim_s$.

(iii) if $(g, g') \in R(\overline{n})$ for some $g' \in G(\overline{n})$, then there is a $\gamma' \in \hat{G}(\overline{m})$ such that $(\gamma, \gamma') \in \hat{R}(\overline{m})$ and $(g', \gamma') \in \sim_s$.

Assume $(g, g') \in R(\overline{n})$ by means of action a. Then, $\zeta_{\overline{n}}(g, a, g')$ is enabled at γ resulting at an abstract state $\gamma' = \delta_{\overline{n}}(g')$ with $(\gamma, \gamma') \in \sim_s$.

□

Following the above result, given a PIIS $\mathcal{S}$ and an m-indexed ACTL*K$\backslash X$ formula $\forall_{\overline{v}}\phi(\overline{v})$, the PIIS procedure checks whether $\forall_{\overline{v}}\phi(\overline{v})$ is satisfied by $\hat{\mathcal{S}}(\overline{m})$. If so, the procedure concludes that $\forall_{\overline{v}}\phi(\overline{v})$ is satisfied in the general case and terminates. Otherwise, if $\forall_{\overline{v}}\phi(\overline{v})$ is not satisfied by $\hat{\mathcal{S}}(\overline{m})$, then PIIS cannot draw any conclusions on the parameterised model checking problem. It thus proceeds to step 2 where it attempts to identify a cutoff.

5.2.3 Step 2: simulation check

With the falsification of a given $\overline{m}$-indexed formula on the abstract system, the PIIS procedure iteratively checks whether there is a $\overline{c} \geq \overline{m}$ such that the concrete system $\mathcal{S}(\overline{c})$ *gs-simulates* the abstract system. If $\mathcal{S}(\overline{c})$ exists, then $\overline{c}$ is a cutoff. The existence of a gs-simulation between the abstract system and a concrete system exists whenever: (i) the concrete system can simulate the actions of the agents in $\mathcal{A}(\overline{m})$ in $\hat{\mathcal{S}}(\overline{m})$; (ii) every agent can always succeed in globally synchronising. Informally, the latter condition is expressed as follows: for every role-synchronous or global-synchronous action performed in $\hat{\mathcal{S}}(\overline{m})$, an agent can reach a template state enabling the action via a sequence of asynchronous and agent-environment actions and without forcing the state of the environment to change. This property not only ensures that an agent can update its state to enable the action, but as the agent communicates only with the environment without forcing the environment's state to change, it also ensures that the agent does not block another agent from global synchronisation.

Given a state l of an agent template, a state l_E of the environment template, and a role-synchronous or a global-synchronous action a, we write $(l, l_E) \rightsquigarrow_a l'$ to mean that: (i) l' is reachable from l via a sequence σ of asynchronous and agent-environment actions; (ii) a is enabled at l'; and (iii) l_E is reachable from itself via the sequence of agent-environment actions appearing in σ. For two abstract states γ, γ', we write $\gamma \rightsquigarrow \gamma'$ to denote that γ' is reachable from γ in zero or more steps via a sequence of actions in which each action is not admitted by an agent in $\mathcal{A}(\overline{m})$. For an action a, we write $\gamma \rightarrow_a \gamma'$ to mean that γ' is reachable from γ in one step by means of action a. $g \rightsquigarrow g'$ and $g \rightarrow_a g'$ are defined for two concrete states g, g' in the same way. We now define the gs-simulation.

Definition 5.6 (gs-simulation)**.** *Given a PIIS $\mathcal{S}$ of k roles and $\overline{m} \in \mathbb{N}^k$, a concrete system $\mathcal{S}(\overline{c})$ with $\overline{c} \geq \overline{m}$ is said to gs-simulate $\hat{\mathcal{S}}(\overline{m})$, denoted $\hat{\mathcal{S}}(\overline{m}) \leq_{gs} \mathcal{S}(\overline{c})$, if there is a relation $\sim_{gs} \subseteq \hat{G}(\overline{m}) \times (L_1 \times \ldots \times L_k) \times G(\overline{c})$ such that $(\hat{\iota}(\overline{m}), (\iota_1, \ldots, \iota_k), \iota(\overline{c})) \in \sim_{gs}$ and whenever $(\gamma, (l_1, \ldots, l_k), g) \in \sim_{gs}$ we have the following:*

1. *$ls_i^j(\gamma) = ls_i^j(g)$, for each agent $(i, j) \in \mathcal{A}(\overline{m})$.*
2. *If $\gamma \rightsquigarrow \gamma^1 \rightarrow_a \gamma^2$, where a is admitted by an agent in $\mathcal{A}(\overline{m})$, then $g \rightsquigarrow g^1 \rightarrow_b g^2$ and the following hold:*
   - *(a) if a is an asynchronous action or an agent-environment action, then:*
     - *$a = b$;*
     - *$(\gamma^2, (l_1, \ldots, l_k), g^2) \in \sim_{gs}$.*
   - *(b) if a is a role-synchronous action from agent template $\mathcal{T}_i$, then:*
     - *$a = b$;*
     - *$(l_i, ls_E(g^1)) \rightsquigarrow_a l_i'$, for some template state l_i';*
     - *$(\gamma^2, (l_1, \ldots, l_i'', \ldots, l_k), g^2) \in \sim_{gs}$, where $l_i'' = t_i(l_i', a)$.*

(c) if a is a global-synchronous action, then:

- $a = b$;
- $(l_1, ls_E(g^1)) \rightsquigarrow_a l'_1, \ldots, (l_k, ls_E(g^1)) \rightsquigarrow_a l'_k$, *for some template states* $l'_1, \ldots, l'_k$;
- $(\gamma^2, (l''_1, \ldots, l''_k), g^2) \in \sim_{gs}$, *where* $l''_1 = t_1(l'_1, a)$, …, $l''_k = t_k(l'_k, a)$.

(d) if a is a multi-role action performed by two agents in $\mathcal{A}(\overline{m})$, then:

- $a = b$;
- $(\gamma^2, (l_1, \ldots, l_k), g^2) \in \sim_{gs}$.

(e) if $a = (x, (l, *), (i, j))$ is a multi-role action performed by an agent $(i, j) \in \mathcal{A}(\overline{m})$ and an abstract agent $\hat{a}_r$, then:

- *b is the instantiation of x shared between the agent (i, j) and any of the agents $(r, \overline{m}.r + 1), \ldots, (r, \overline{c}.r)$;*
- $(\gamma^2, (l_1, \ldots, l_k), g^2) \in \sim_{gs}$.

Condition 1 insists on the equality of the local states of the agents in $\mathcal{A}(\overline{m})$. Conditions 2(a), 2(b), and 2(e) require the simulation of the actions performed by the agents in $\mathcal{A}(\overline{m})$: whenever the abstract system can reach a state enabling an action admitted by an agent in $\mathcal{A}(\overline{m})$, the concrete system can reach a state enabling the same action for the agent. Additionally, conditions 2(c) and 2(d) require that whenever said action is a role-synchronous or a global-synchronous action, then a concrete agent can move to a state enabling the action by means of asynchronous and agent-environment actions and without altering the state of the environment.

5.2.4 Summary

The PIIS procedure is given by Algorithm 2. Given a PIIS $\mathcal{S}$ an $\overline{m}$-indexed ACTL*K$\setminus X$ formula $\forall_{\overline{v}} \phi(\overline{v})$, the procedure constructs the abstract model $\hat{\mathcal{S}}(\overline{m})$. It then checks $\hat{\mathcal{S}}(\overline{m})$ against the trivial instantiation $\phi[trivial]$ of $\forall_{\overline{v}} \phi(\overline{v})$. If $\hat{\mathcal{S}}(\overline{m}) \models \phi[trivial]$, then PIIS returns $true$. Otherwise, PIIS iteratively checks for a concrete system $\mathcal{S}(\overline{c})$ such that $\mathcal{S}(\overline{c})$ gs-simulates $\hat{\mathcal{S}}(\overline{m})$. $\mathcal{S}(\overline{c})$ is not generally guaranteed to exist. If it does exist, then $\mathcal{S}(\overline{c})$ is the cutoff system. Following the cutoff identification, PIIS checks the set $\{\mathcal{S}(\overline{x}) \colon \overline{m} \leq \overline{x} \leq \overline{c}\}$ of concrete systems against the

Algorithm 2 Parameterised model checking procedure for parameterised interleaved interpreted systems.

```
procedure PIIS($\mathcal{S}$,$\forall_{\overline{v}}\phi(\overline{v})$)
    build $\hat{\mathcal{S}}(\overline{m})$
    if $\hat{\mathcal{S}}(\overline{m}) \models \phi[trivial]$ then
        return $true$;
    end if
    $\bar{c} \leftarrow \bar{m}$
    do
        increase $\bar{c}$ in a breadth-first manner
    until $\hat{\mathcal{S}}(\overline{m}) \ \leq_{gs} \ \mathcal{S}(\bar{c})$
    for all $\bar{x}$ such that $\bar{m} \leq \bar{x} \leq \bar{c}$ do
        if $\mathcal{S}(\bar{x}) \not\models \phi[trivial]$ then
            return $false$;
        end if
    end for
    return $true$;
end procedure
```

trivial instantiation $\phi[trivial]$ of $\forall_{\overline{v}}\phi(\overline{v})$. If $\phi[trivial]$ is not satisfied by at least one system, then the procedure returns *false*, otherwise it returns *true*.

This concludes the description of the PIIS procedure. Algorithm 1 provides a methodology for solving the parameterised model checking problem by giving the conditions under which the problem can be solved by checking each concrete system up to the cutoff system. We assess the soundness of PIIS in the next section.

5.2.5 Proof of soundness

Theorem 5.1. *Let $\mathcal{S}$ be a parameterised interleaved interpreted system and $\forall_{\overline{v}}\phi(\overline{v})$ an $\overline{m}$-indexed ACTL*K\X formula. Then, the following hold:*

1. *If PIIS($\mathcal{S}$, $\forall_{\overline{v}}\phi(\overline{v})$) returns true, then $\mathcal{S} \models \forall_{\overline{v}}\phi(\overline{v})$.*

2. *If PIIS($\mathcal{S}$, $\forall_{\overline{v}}\phi(\overline{v})$) returns $false$, then $\mathcal{S} \not\models \forall_{\overline{v}}\phi(\overline{v})$.*

The second clause follows trivially from the definition of PIIS. For the first clause assume that PIIS($\mathcal{S}$, $\forall_{\overline{v}}\phi(\overline{v})$) returns *true*. Then it is either the case that: (a) $\hat{\mathcal{S}}(\overline{m}) \models \forall_{\overline{v}}\phi(\overline{v})$; or (b) $\forall \overline{m} \leq \overline{x} \leq \overline{c}.\, \mathcal{S}(\overline{x}) \models \forall_{\overline{v}}\phi(\overline{v})$, where $\overline{c} \geq \overline{m}$ with $\hat{\mathcal{S}}(\overline{m}) \leq_{gs} \mathcal{S}(\overline{c})$.

Consider the case (a). By Lemma 5.1, $\hat{\mathcal{S}}(\overline{m})$ simulates every concrete system $\mathcal{S}(\overline{n})$ with $\overline{n} \geq \overline{m}$. Therefore, by Theorem 2.1, we have that $\mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v})$ for every concrete system $\mathcal{S}(\overline{n})$ with $\overline{n} \geq \overline{m}$. It follows that if $\hat{\mathcal{S}}(\overline{m}) \models \forall_{\overline{v}}\phi(\overline{v})$, then $\mathcal{S} \models \forall_{\overline{v}}\phi(\overline{v})$.

Consider the case (b). It suffices to show that if $\hat{\mathcal{S}}(\overline{m}) \leq_{gs} \mathcal{S}(\overline{c})$, then $\mathcal{S}(\overline{c})$ is a cutoff system. The proof is given in two parts. In the first part, we show that $\mathcal{S}(\overline{c})$ $\overline{m}$-stuttering simulates every bigger system. By Theorem 3.1, we then get the following:

$$\mathcal{S}(\overline{c}) \models \forall_{\overline{v}}\phi(\overline{v}) \text{ implies } \mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v}), \text{ for any } \overline{n} \geq \overline{c} \tag{1}$$

In the second part, we show that every bigger system $\overline{m}$-stuttering simulates $\mathcal{S}(\overline{c})$. By Theorem 3.1, we then obtain the following:

$$\mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v}) \text{ implies } \mathcal{S}(\overline{c}) \models \forall_{\overline{v}}\phi(\overline{v}), \text{ for any } \overline{n} \geq \overline{c} \tag{2}$$

From (1) and (2) it follows that

$$\forall \overline{m} \leq \overline{x} \leq \overline{c}.\, \mathcal{S}(\overline{x}) \models \forall_{\overline{v}}\phi(\overline{v}) \text{ iff } \forall \overline{n} \geq \overline{c}.\, \mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v})$$

Therefore, $\mathcal{S}(\overline{c})$ is a cutoff system.

Lemma 5.2 concludes part 1 of the proof and Lemma 5.3 establishes part 2 of the proof.

Lemma 5.2. *Let $\mathcal{S}$ be a parameterised interleaved interpreted system and $\forall_{\overline{v}}\phi(\overline{v})$ an $\overline{m}$-indexed ACTL*K\X formula. Then, $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{c})$, where $\hat{\mathcal{S}}(\overline{m}) \leq_{gs} \mathcal{S}(\overline{c})$ and $\overline{n} \geq \overline{c}$.*

Proof. By Lemma 5.1, $\hat{\mathcal{S}}(\overline{m})$ simulates every system $\mathcal{S}(\overline{n})$ with $\overline{n} \geq \overline{m}$. Since simulation is stronger than $\overline{m}$-stuttering simulation, $\hat{\mathcal{S}}(\overline{m})$ $\overline{m}$-stuttering simulates every system $\mathcal{S}(\overline{n})$ with $\overline{n} \geq \overline{m}$. By assumption, $\mathcal{S}(\overline{c})$ gs-simulates $\hat{\mathcal{S}}(\overline{m})$. Since gs-simulation is stronger than $\overline{m}$-stuttering simulation, $\mathcal{S}(\overline{c})$ $\overline{m}$-stuttering simulates $\hat{\mathcal{S}}(\overline{m})$. From the transitivity of $\overline{m}$-stuttering simulation, $\mathcal{S}(\overline{c})$ $\overline{m}$-stuttering simulates every concrete system $\mathcal{S}(\overline{n})$ with $\overline{n} \geq \overline{c}$. □

Lemma 5.3. *Let $\mathcal{S}$ be a parameterised interleaved interpreted system of k roles and $\forall_{\overline{v}}\phi(\overline{v})$ an $\overline{m}$-indexed ACTL*K\X formula. Then, $\mathcal{S}(\overline{c}) \leq_{\overline{m}ss} \mathcal{S}(\overline{n})$, where $\hat{\mathcal{S}}(\overline{m}) \leq_{gs} \mathcal{S}(\overline{c})$ and $\overline{n} \geq \overline{c}$.*

Proof. We prove a stronger result, i.e., that for every $\overline{n} \geq \overline{c}$, $\mathcal{S}(\overline{n})$ gs-simulates $\hat{\mathcal{S}}(\overline{m})$. Since gs-simulation is stronger than $\overline{m}$-stuttering simulation, it then follows that $\mathcal{S}(\overline{n})$ $\overline{m}$-stuttering simulates $\hat{\mathcal{S}}(\overline{m})$. From Lemma 5.1, $\hat{\mathcal{S}}(\overline{m})$ simulates $\mathcal{S}(\overline{c})$. Since simulation is stronger than $\overline{m}$-stuttering simulation, $\hat{\mathcal{S}}(\overline{m})$ $\overline{m}$-stuttering simulates $\mathcal{S}(\overline{c})$. Therefore, from the transitivity of $\overline{m}$-stuttering simulation, if $\mathcal{S}(\overline{n})$ gs-simulates $\hat{\mathcal{S}}(\overline{m})$, then $\mathcal{S}(\overline{n})$ $\overline{m}$-stuttering simulates $\mathcal{S}(\overline{c})$.

Let $\overline{n} \geq \overline{c}$. We now show that $\mathcal{S}(\overline{n})$ gs-simulates $\hat{\mathcal{S}}(\overline{m})$. Let Q^c be the gs-simulation between $\hat{\mathcal{S}}(\overline{m})$ and $\mathcal{S}(\overline{c})$. We start by defining the relation $Q^n \subseteq \hat{G}(\overline{m}) \times (L_1 \times \ldots \times L_k) \times G(\overline{n})$ between the abstract states in $\hat{\mathcal{S}}(\overline{m})$, the Cartesian product of the sets of template states for the agent templates, and the concrete states in $\mathcal{S}(\overline{n})$. A tuple $(\gamma, (l_1, \ldots, l_k), g)$ is in Q^n if the tuple resulting from collapsing g to the local components of the agents in $\mathcal{A}(\overline{c})$ is in Q^c:

$$(\gamma, (l_1, \ldots, l_k), g) \in Q^n \text{ if } \Big(\gamma, (l_1, \ldots, l_k)\,, \Big(ls_1^1(g), \ldots, ls_k^{\overline{c}.k}(g)\Big)\Big) \in Q^c.$$

We show that Q^n is a gs-simulation relation between $\hat{\mathcal{S}}(\overline{m})$ and $\mathcal{S}(\overline{n})$. It should be clear that $(\hat{\iota}(\overline{m}), (\iota_1, \ldots, \iota_k), \iota(\overline{n})) \in Q^n$, where $\hat{\iota}(\overline{m}), \iota(\overline{n})$ are the initial states of $\hat{\mathcal{S}}(\overline{m})$ and $\mathcal{S}(\overline{n})$, respectively. Let $(\gamma, (l_1, \ldots, l_k), g) \in Q^n$ be arbitrary. We have to show that the gs-simulation requirements 1 and 2(a)-2(e) are satisfied.

1. $ls_i^j(\gamma) = ls_i^j(g)$, for each $(i, j) \in \mathcal{A}(\overline{m})$.

   Simulation requirement 1 follows trivially from the definition of Q^n.

2. If $\gamma \rightsquigarrow \gamma^1 \rightarrow_a \gamma^2$, where a is admitted by an agent in $\mathcal{A}(\overline{m})$, then $g \rightsquigarrow g^1 \rightarrow_b g^2$ and the following hold:

   (a) if a is an asynchronous action or an agent-environment action, then $a = b$ and $(\gamma^2, (l_1, \ldots, l_k), g^2) \in \sim_{gs}$.
   Assume that $\gamma \rightsquigarrow \gamma^1 \rightarrow_a \gamma^2$, where a is admitted by an agent in $\mathcal{A}(\overline{m})$. Let $g' = \big(ls_1^1(g), \ldots, ls_k^{\overline{c}.k}(g)\big)$. Since $(\gamma, (l_1, \ldots, l_k)\,, g') \in Q^c$, we have that $g' \rightsquigarrow g'^1 \rightarrow_a g'^2$ and $(\gamma^2, (l_1, \ldots, l_k), g'^2) \in Q^c$. Assume σ to be the sequence of actions performed in the path from g' to g'^1. As σ does not contain any role-synchronous or global-synchronous actions, we have that $g \rightsquigarrow g^1 \rightarrow_a g^2$ by means of the sequence σ, and $(\gamma^2, (l_1, \ldots, l_k), g^2) \in Q^n$.

(b) if a is a role-synchronous action from agent template $\mathcal{T}_i$, then $a = b$, $(l_i, ls_E(g^1)) \rightsquigarrow_a l'_i$ for some template state l'_i, and $(\gamma^2, (l_1, \dots, l''_i, \dots, l_k), g^2) \in \sim_{gs}$, where $l''_i = t_i(l'_i, a)$.

Assume that $\gamma \rightsquigarrow \gamma^1 \rightarrow_a \gamma^2$, where a is a role-synchronous action from agent template i. Let $g' = \left(ls^1_1(g), \dots, ls^{\overline{c}.k}_k(g)\right)$. Since $(\gamma, (l_1, \dots, l_k), g') \in Q^c$, the following hold: (i) $g' \rightsquigarrow g'^1 \rightarrow_a g'^2$; (ii) $(l_i, ls_E(g'^1))) \rightsquigarrow_a l'_i$ for some template state l'_i; and (iii) $(\gamma^2, (l_1, \dots, l''_i, \dots, l_k), g'^2) \in Q^c$, where $l''_i = t_i(l'_i, a)$.

Assume σ to be the sequence of actions performed in the path from g' to g'^1. Consider $a_1 \dots a_x$ to be the sequence of template asynchronous and template agent-environment actions appearing in the template path from l_i to l'_i. Extend σ to the sequence of actions $\sigma' = \sigma \circ (a_1)^1_i \dots (a_x)^1_i \dots (a_1)^{\overline{n}.i}_i \dots (a_x)^{\overline{n}.i}_i$, where the actions $(a_1)^j_i \dots (a_x)^j_i$ are the concrete instantiations of $(a_1 \dots a_x)$, respectively, for agent template (i, j). It follows that $g \rightsquigarrow g^1 \rightarrow_a g^2$ by means of the sequence σ'. Additionally, it should be clear that $(l_i, ls_E(g^1))) \rightsquigarrow_a l'_i$ and $(\gamma^2, (l_1, \dots, l''_i, \dots, l_k), g^2) \in Q^n$, where $l''_i = t_i(l'_i, a)$.

(c) if a is a global-synchronous action, then $a = b$, $(l_1, ls_E(g^1)) \rightsquigarrow_a l'_1, \dots, (l_k, ls_E(g^1)) \rightsquigarrow_a l'_k$ for some template states $l'_1, \dots, l'_k$, and $(\gamma^2, (l''_1, \dots, l''_k), g^2) \in \sim_{gs}$, where $l''_1 = t_1(l'_1, a), \dots, l''_k = t_k(l'_k, a)$.

Simulation requirement 2(c) is similarly proven to simulation requirement 2(b) by extending σ for each agent template.

(d) if a is a multi-role action performed by two agents in $\mathcal{A}(\overline{m})$, then $a = b$, and $(\gamma^2, (l_1, \dots, l_k), g^2) \in \sim_{gs}$.

The proof is identical to the proof given for 2(a).

(e) if $a = (x, (l, *), (i, j))$ is a multi-role action performed by an agent $(i, j) \in \mathcal{A}(\overline{m})$ and an abstract agent $\hat{a}_r$, then: (i) b is the instantiation of x shared between the agent (i, j) and any of the agents $(r, \overline{m}.r{+}1), \dots, (r, \overline{c}.r)$; and (ii) $(\gamma^2, (l_1, \dots, l_k), g^2) \in \sim_{gs}$.

The proof is identical to the proof given for 2(a).

We have thus proven that the gs-simulation requirements are satisfied for an arbitrary tuple $(\gamma, (l_1, \dots, l_k), g)$ in Q^n. Therefore, Q^n is a gs-simulation between $\hat{\mathcal{S}}(\overline{m})$ and $\mathcal{S}(\overline{n})$. Hence, $\mathcal{S}(\overline{n})$ $\overline{m}$-stuttering simulates $\mathcal{S}(\overline{c})$. $\square$

5.3 Applications

5.3.1 Train-Gate-Controller

In section 3.2.1 we represented the prioritised variant of the Train-Gate-Controller as a parameterised interleaved interpreted system $\mathcal{S}_{TGC}$ composed of an agent template PT representing prioritised trains, an agent template NT representing normal trains, and an environment template representing the controller. In section 3.3 we encoded the property "whenever a train is in the tunnel, it knows that no other train is in the tunnel at the same time" in the following indexed ACTL*K$\backslash X$ formula:

$$\begin{aligned}\phi_{TGC} = \forall_{(\{u,v\},\{x,y\})} AG \, (((pt,u) \rightarrow K^{u}_{PT} \, (\neg(pt,v) \wedge \neg(nt,x))) \\ \wedge \, ((nt,x) \rightarrow K^{x}_{NT} \, (\neg(nt,y) \wedge \neg(pt,u)))) \, ,\end{aligned}$$

where u, v are variables of PT, x, y are variables of NT, the atomic proposition pt holds in the template states in which the template prioritised train is in the tunnel, and the atomic proposition nt holds in the template states in which the template normal train is in the tunnel.

We now analyse the above formula using the PIIS procedure. We first observe that ϕ_{TGC} is a $(2,2)$-indexed formula. Therefore, in step 1, we build the abstract model $\hat{\mathcal{S}}_{TGC}\,((2,2))$. A fragment of $\hat{\mathcal{S}}_{TGC}\,((1,1))$ is depicted in Figure 5.1; $\hat{\mathcal{S}}_{TGC}\,((2,2))$ is similarly obtained by considering two concrete agents for each type of train. In step 2, we search for a concrete system that gs-simulates $\hat{\mathcal{S}}_{TGC}\,((2,2))$. It can be established (see Chapter 7) that $\hat{\mathcal{S}}_{TGC}\,((2,2)) \leq_{gs} \mathcal{S}\,((3,3))$. We can thus assess the correctness of $\mathcal{S}_{TGC}$ by checking the systems $\mathcal{S}_{TGC}\,((2,2))$, $\mathcal{S}_{TGC}\,((3,2))$, $\mathcal{S}_{TGC}\,((2,3))$, and $\mathcal{S}_{TGC}\,((3,3))$. These checks can be performed by any epistemic model checker, which would return $true$, thereby establishing the correctness of $\mathcal{S}_{TGC}$ irrespectively of the number of trains in the system.

5.3.2 The Alpha swarm aggregation algorithm

This section analyses the Alpha swarm aggregation algorithm [WLNM08] which is a simpler protocol than the Beta algorithm discussed in section 4.3.2. The analysis is built on the setting discussed in this section 4.3.2. In particular, we assume that each robot operates on a two-dimensional arena and communicates with its peers and the environment by means of a wireless sensor of limited range. A robot is said to be in another robot's neighbourhood if the position of the former is in the range of the latter's sensor. The arena is assumed to be finite and to wrap around. We assume that the robots update their state with high frequency and model them by assuming synchronicity.

According to the Alpha algorithm, each robot i follows the following protocol. Through local communication the robot observes the number $N(i)$ of neighbouring robots. The robot is said to be connected if its neighbourhood is composed of at least α robots, for a threshold α; i.e., $N(i) \geq \alpha$. The behaviour of each robot is characterised by its connectivity status and by whether it is in *forward (motion) mode* or in *coherence (motion) mode*. More specifically, we have the following:

- if a robot is in forward mode and connected, then it moves forward;
- if it is in forward mode, but not connected, then it performs a 180° turn and changes its motion mode to 'coherence';
- if it is in coherence mode, but not connected, then it moves forward;
- if it is in coherence mode and connected, then it performs a random 90° turn and changes its motion mode to 'forward'.

We can encode the Alpha algorithm as a PIIS $\mathcal{S}_\alpha$ consisting of an agent template R_α and an environment template $\mathcal{E}$. Figure 5.2 depicts a fragment of the encoding for a 2×2 arena and a wireless range of 1.

The states of the agent template are given as tuples of five components: (i) the position on the grid; (ii) the direction of movement $(North, East, South, West)$; (iii) the motion mode $(Forward, Coherence)$; (iv) the connectivity status $(Connected, \neg Connected)$; (v) the phase $(p1, p2, p3a, p3b)$ of the encoding. The encoding is in terms of three phases. For each phase $p1, p2, p3a, p3b$ an action is performed by each of the robots. The actions that can be performed in each of the phases are described below. All the actions update the phase component of a robot to either $s1$, $s2$, $s3$, or $s3b$, depending on whether the robot is currently in phase $p1$, $p2$, $p3$, or $p3b$, respectively. In each of the $s1$, $s2$, $s3$, $s3b$ phases only the global synchronous action syn is enabled for each of the robots. The action updates the phase component of each of the robots to the next phase in a round-robin fashion; in other words, the syn action simulates synchronisity in the agents' actions. Initially, the robots are in square $(1, 1)$, they do not have a direction of movement, they are connected, and they are in phase $p1$.

- In phase $p1$, each robot performs one of the asynchronous actions $north, east, south, west$, thereby choosing an initial random direction.
- In phase $p2$, each robot moves forward one step by performing the asynchronous action $move$.
- Phase $p3$ is responsible for updating the connectivity status of each of the robots. This is done in 2 steps:
  1. In phase $p3a$, each robot performs the agent-environment action (x, y), where (x, y) is the location of the robot. A template environment's state has a component $((x, y), z)$, where $z \in \{1, \ldots, \alpha + 1\}$, for each square (x, y) in the grid. The environment increases z, up to $\alpha + 1$, each time an (x, y) action is performed. Following this, the environment can deduce whether or not a robot in (x, y) is connected.
  2. In phase $p3b$, for each location (x, y), the environment's protocol enables the agent-environment action $con\_(x, y)$ if the sum of z's in the range of (x, y) is at least $\alpha + 1$, otherwise it enables the agent-environment action $\neg con\_(x, y)$. Thus, each robot can update its connectivity status by synchronising with the environment through the con actions. This happens for all robots in a sequence of steps before the system may move to the next phase; upon each synchronisation, the corresponding agent updates its direction and motion mode as described above. Then, the system goes back to phase $p2$ where the counters in the environment are reset to zero in repeating the cycle.

We now use the PIIS procedure to analyse an instance of the Alpha algorithm where the sensor range is equal to 1, the alpha parameter is equal to 2, and the arena is of size 5×5. This instance is of particular interest given its failure to satisfy the connectedness property when 3 robots constitute the swarm [DWFZ12]. The property is given by the following 1-indexed formula

$$\phi_\alpha = \forall_{\{v\}} K_v GF(con, v)$$

expressing "each robot knows that it will be infinitely often connected". Given the nature of swarms, however, an emergent behaviour may be exhibited when additional agents are present and may continue to be exhibited in any bigger system. Thus, we proceed to investigate whether the connectedness property is satisfied in the presence of additional agents.

We first establish (see Chapter 7) that the abstract system $\hat{\mathcal{S}}_\alpha(1)$ does not satisfy ϕ_α. By Lemma 5.1, $\hat{\mathcal{S}}_\alpha$ is able to simulate the concrete system of 3 agents, where, as described in [DWFZ12], 2 agents may initially go East and the remaining agent may initially go West. In this case the pair of agents initially going East are afterwards always connected, in forward mode, and moving East, whereas the agent initially going West is afterwards never connected, in coherence mode, and moving East. Following the falsification of ϕ_α on the abstract model, it can be established that the concrete system with 3 agents gs-simulates the abstract system. Therefore, $\mathcal{S}_\alpha(3)$ is a cutoff. Using any epistemic model checker, it can be established that $\mathcal{S}_\alpha(3) \not\models \phi_\alpha$. This allows us to conclude that the connectedness property is not satisfied by the alpha algorithm irrespective of the number of robots in the swarm.

$$\begin{tikzcd}
{[\mathbf{11},-,\mathbf{F},\mathbf{C},\mathbf{p1}]} \arrow[r, "north"] \arrow[d, dotted, no head, "{south, east \; west}"'] & {[\mathbf{11},\mathbf{N},\mathbf{F},\mathbf{C},\mathbf{s1}]} \arrow[r, "syn"] & {[\mathbf{11},\mathbf{N},\mathbf{F},\mathbf{C},\mathbf{p2}]} \arrow[r, "move"] & {[\mathbf{12},\mathbf{N},\mathbf{F},\mathbf{C},\mathbf{s2}]} \arrow[r, "syn"] \arrow[d, dotted, no head, "con\_12"] & {[\mathbf{12},\mathbf{N},\mathbf{F},\mathbf{C},\mathbf{p3a}]} \arrow[d, "12"] \\
\cdots\cdots \arrow[r, dotted, no head] & {[\mathbf{12},\mathbf{S},\mathbf{C},\neg\mathbf{C},\mathbf{p2}]} & {[\mathbf{12},\mathbf{S},\mathbf{C},\neg\mathbf{C},\mathbf{s3b}]} \arrow[l, "syn"'] & {[\mathbf{12},\mathbf{N},\mathbf{F},\mathbf{C},\mathbf{p3b}]} \arrow[l, "\neg con\_12"'] & {[\mathbf{12},\mathbf{N},\mathbf{F},\mathbf{C},\mathbf{s3a}]} \arrow[l, "syn"']
\end{tikzcd}$$

(a) Fragment of the agent template of the Alpha algorithm.

$$\begin{tikzcd}
& & {} & {} \\
{[\mathbf{110},\mathbf{120},\mathbf{210},\mathbf{220},\mathbf{p1}]} \arrow[r, "syn"] & {[\mathbf{110},\mathbf{120},\mathbf{210},\mathbf{220},\mathbf{p2}]} \arrow[r, "syn"] & {[\mathbf{110},\mathbf{120},\mathbf{210},\mathbf{220},\mathbf{p3a}]} \arrow[r, "12"] \arrow[u, dotted, no head, "{11, 21, 22, syn}"'] & {[\mathbf{110},\mathbf{121},\mathbf{210},\mathbf{220},\mathbf{p3a}]} \arrow[d, "21"] \arrow[u, dotted, no head, "{11, 12, 22, syn}"'] \\
& {[\mathbf{110},\mathbf{121},\mathbf{212},\mathbf{220},\mathbf{p3b}]} \arrow[u, "syn"] \arrow[loop below, "{\neg con\_11, \neg con\_12, con\_21, \neg con\_22}"] & {[\mathbf{110},\mathbf{121},\mathbf{212},\mathbf{220},\mathbf{p3a}]} \arrow[l, "syn"'] \arrow[loop above, "21"] \arrow[d, dotted, no head, "{11, 12, 22}"] & {[\mathbf{110},\mathbf{121},\mathbf{211},\mathbf{220},\mathbf{p3a}]} \arrow[l, "21"'] \arrow[d, dotted, no head, "{11, 12, 22, syn}"] \\
& & {} & {}
\end{tikzcd}$$

(b) Fragment of the environment template of the Alpha algorithm.

Figure 5.2: Fragment of the parameterised interleaved interpreted system of the Alpha algotithm.

Chapter 6

The $\mathbb{SMR}$, $\mathbb{SGS}$ and $\mathbb{SFE}$ classes of PIIS

Because of their importance with respect to their amenability to verification, this chapter identifies three noteworthy classes of PIIS. The classes correspond to different combinations of template actions. For each of the classes a parameterised model checking procedure is presented. The procedures are used to verify examples from the MAS literature in a parametric setting, including the robot foraging scenario [Liu07], the Train-Gate-Controller [HW02b], and the autonomous robot example [FHMV95]. Their crucial difference from the `PIIS` procedure is a polynomial cutoff identification methodology instead of an exponential one. Section 6.1 fixes the notation and introduces the classes. Sections 6.2, 6.3, and 6.4 study these classes in detail.

6.1 Introduction

We outline parameterised interleaved interpreted systems as presented in Section 3.2, we fix the chapter's notation, and we define the $\mathbb{SMR}$, $\mathbb{SGS}$, and $\mathbb{SFE}$ classes of PIIS.

A parameterised interleaved interpreted system is a tuple

$$PIIS = (\mathcal{T}, \mathcal{E}, \mathcal{V}),$$

where $\mathcal{T} = \{\mathcal{T}_i = (L_i, \iota_i, Act_i, P_i, t_i) \colon 1 \leq i \leq k\}$ is a set of $k \geq 1$ agent templates, $\mathcal{E} =$

$(L_{\mathcal{E}}, \iota_{\mathcal{E}}, Act_{\mathcal{E}}, P_{\mathcal{E}}, t_{\mathcal{E}})$ is an environment template, and $\mathcal{V} = \{V_i : L_i \to \mathcal{P}(AP_i) \colon 1 \leq i \leq k\}$ is a set of valuation functions on the template states.

Agents from template $\mathcal{T}_i$ may evolve asynchronously via asynchronous A_i actions, communicate with the environment via agent-environment AE_i actions, synchronise with the agents of the same role via role-synchronous RS_i actions, synchronise with all the agents in the system via global-synchronous GS actions, and communicate with an agent performing another role via multi-role MR_i actions. We write $l \dashrightarrow_a l'$ to mean that $t_i(l,a) = l'$. Given a set of template actions $X \subseteq Act_i$ we let $l \dashrightarrow_X l'$ denote that $l \dashrightarrow_a l'$ for some $a \in X$. The reflexive and transitive closure of $\dashrightarrow_X$ is denoted by $\dashrightarrow_{X*}$.

Given a concrete value $\overline{n} = (n_1, \ldots, n_k)$ of the system's parameter, the concrete system $PIIS(\overline{n})$ is the interleaved interpreted system

$$PIIS(\overline{n}) = \Big((L_i^j, \iota_i^j, Act_i^j, P_i^j, t_i^j)_{(i,j)\mathcal{A}(\overline{n})}, (L_E, \iota_E, Act_E, P_E, t_E), \iota, V\Big).$$

$PIIS(\overline{n})$ results from the composition of $\overline{n}.i$ instantiations $\{(i,1), \ldots, (i, \overline{n}.i)\}$ of each agent template i and an instantiation of the environment template. For each concrete system $PIIS(\overline{n})$ we associate a temporal-epistemic model

$$\mathcal{S}_{PIIS(\overline{n})} = \Big(G(\overline{n}), \iota(\overline{n}), R(\overline{n}), (\mathcal{K}_i^j)_{(i,j)\in\mathcal{A}(\overline{n})}, V(\overline{n})\Big)$$

that can be used to interpret temporal-epistemic formulae. When $PIIS(\overline{n})$ is clear from the context we simply write $\mathcal{S}(\overline{n})$ for $\mathcal{S}_{PIIS(\overline{n})}$. We assume that the joint null action is always enabled in a concrete system. Therefore, the concrete transition relations are serial.

Given a global state g, we write $ls_i^g(g)$ to denote the local state of agent (i,j) in g. By $tls_i^j(g)$, we express the template state from which the local state of agent (i,j) in g has been instantiated. We write $g \to_a g'$ to denote that $(g, g') \in \mathcal{R}(\overline{n})$ by means of action a. For any set of concrete actions X, $g \to_X g'$ expresses that $g \to_a g'$ for an action $a \in X$. The reflexive and transitive closure of $\to_X$ is denoted by $\to_{X*}$.

A path π is a sequence $\pi = g^1 a^1 g^2 a^2 g^3 \ldots$ such that $g^i \to_{a^i} g^{i+1}$, for every $i \geq 1$. Given a path π, we write $\pi(i)$ ($\pi(i, Act)$, respectively) for the i-th state (action, respectively) in π. If π is finite, then we write $\pi[]$ for the last state in π. By $\pi[i]$, we denote the suffix $g^i a^i g^{i+1} \ldots$ of π, and by $[i]\pi$ we denote its prefix $g^1 a^1 \ldots g^i$. The set of all paths originating from a state g is denoted by $\Pi(g)$. Since the global transition relation is deterministic we sometimes (uniquely) denote a path $g^1 a^1 g^2 a^2 \ldots$ by the sequence $g^1 a^1 a^2 \ldots$.

We now define the $\mathbb{SMR}$, $\mathbb{SGS}$, and $\mathbb{SFE}$ classes.

1. $\mathbb{SMR} = \left\{ \mathcal{S} \colon \mathcal{S} \text{ is a PIIS composed of } k \geq 1 \text{ roles such that } \bigcup_{1 \leq i \leq k} RS_i \cup GS = \emptyset \right\}$.

   $\mathbb{SMR}$ is the class of PIIS generated from agent templates defined only on asynchronous, agent-environment, and *guarded* multi-role actions. A multi-role action is said to be guarded for an agent template $\mathcal{T}_i$ if the template transition of the template is such that it returns the same template state at which the action is performed. A concrete multi-role action is said to be guarded by a concrete agent (i, r) if the action is instantiated from an action that is guarded by $\mathcal{T}_i$. In a similar manner to the model of disjunctive guards [EK00], whenever a concrete multi-role action that is guarded by a concrete agent is performed in a global transition, said agent remains in its current local state.

   We have thus far considered a set $MR_{i,j} = MR_{j,i}$ of template multi-role actions shared between the templates $\mathcal{T}_i$ and $\mathcal{T}_j$, for each pair $\mathcal{T}_i$,$\mathcal{T}_j$ of agent templates. For the $\mathbb{SMR}$ class of systems we insist on each multi-role action to be guarded by precisely one of the two templates sharing the action. We write $MR_{i,j}$ for the set of multi-role actions that are shared between templates $\mathcal{T}_i$, $\mathcal{T}_j$, and guarded by $\mathcal{T}_j$; by $MR_{j,i}$, we mean the set of multi-role actions that are shared between templates $\mathcal{T}_i$, $\mathcal{T}_j$, and guarded by $\mathcal{T}_i$. We write $(a, (i, r), (\mathbf{j}, \mathbf{q}))$ to denote the instantiation of a template multi-role action $a \in MR_{i,r}$ that is shared between the agents (i, j), (r, q), and guarded by (j, q). By MR_i^j, we denote the set of concrete multi-role actions of agent (i, j) that are not guarded by (i, j).

   Decentralised systems and security protocols may be encoded in $\mathbb{SMR}$ using the machinery of multi-role actions, whereas centralised systems can be represented in $\mathbb{SMR}$ using the communication primitive of agent-environment actions. As a result, the $\mathbb{SMR}$ class is particularly suitable for modelling swarm robotics, which are naturally decentralised systems, but interacting with their environment [BDT99].

2. $\mathbb{SGS} = \left\{ \mathcal{S} \colon \mathcal{S} \text{ is a PIIS composed of } k \geq 1 \text{ roles such that } \bigcup_{1 \leq i \leq k} (RS_i \cup MR_i) = \emptyset \right\}$.

$\mathbb{SGS}$ is the class of PIIS generated from agent templates defined only on asynchronous, agent-environment, and global-synchronous actions. This class can represent broadcast protocols, cache coherence protocols, swarm aggregation algorithms in a grid environment, and several scenarios where synchronous handshaking is required.

3. $\mathbb{SFE} = \left\{\mathcal{S} \colon \mathcal{S} \text{ is a PIIS composed of } k \geq 1 \text{ roles such that } \bigcup_{1\leq i\leq k}(AE_i \cup MR_i) = \emptyset\right\}$.

   $\mathbb{SFE}$ is the class of PIIS generated from agent templates defined only on asynchronous, role-synchronous, and global-synchronous actions. The absence of agent-environment and multi-role actions implies that all the agents evolve in the same way following synchronisation with the environment. Differently from the $\mathbb{SMR}$ and $\mathbb{SGS}$ classes, the parameterised model checking problem for this class is, as we will show, decidable. This gives clear advantages when protocols can be expressed by $\mathbb{SFE}$.

An example of an $\mathbb{SMR}$ system is the robot foraging scenario discussed in section 3.2.1, an example of an $\mathbb{SGS}$ system is the Train-Gate-Controller described in section 3.2.1, an example of an $\mathbb{SFE}$ system is the autonomous robot example given in section 3.2.1. We study the $\mathbb{SMR}$, $\mathbb{SGS}$, and $\mathbb{SFE}$ classes in detail in sections 6.2, 6.3, and 6.4, respectively.

6.2 Verifying $\mathbb{SMR}$ systems

We begin with the $\mathbb{SMR}$ class of systems defined on asynchronous, agent-environment and guarded multi-role actions. We identify a notion of simulation and show that the existence of this simulation between the agent and environment templates guarantees a cutoff, which we show how to calculate given a PIIS and a specification. This will enable us to define a model checking procedure which we will exemplify on the robot foraging scenario following a discussion on its applicability.

6.2.1 Agent-environment simulation

We fix a PIIS $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V})$ composed of k agent templates and an environment template. To simplify our analysis, in the following we assume, without loss of generality, that for each

$$\begin{tikzcd}
\mathbf{l_i} \arrow[r, dashed, "A_i*"] \arrow[d, dotted, dash, "\sim_{aes}"'] & \mathbf{l'_i} \arrow[r, dashed, "a \in AE_i \cup MR_i \cup GS"] & \mathbf{l''_i} \arrow[d, dotted, dash, "\sim_{aes}"] \\
\mathbf{l_E} \arrow[rr, dashed, "a"] & & \mathbf{l'_E}
\end{tikzcd}$$

Figure 6.1: Agent-environment simulation between $\mathcal{T}_i$ and $\mathcal{E}$.

action $a \in Act_{\mathcal{E}}$ we have that $|\{l \in L_{\mathcal{E}} \colon a \in P_{\mathcal{E}}(l)\}| = 1$; i.e., an environment's action is enabled at exactly one template state. Note that any PIIS can be translated into a PIIS for which each action of the environment is enabled at exactly one template state. The translation is given in two steps.

1. Replace each template transition $l_E \dashrightarrow_a l'_E$ of the environment template with $l_E \dashrightarrow_{(l_E,a)} l'_E$, where (l_E, a) is a fresh action of the same type with a.

2. Replace each template transition $l \dashrightarrow_a l'$ of an agent template with the set of transitions $X = \big\{ l \dashrightarrow_{(l_E,a)} l' \colon (l_E, a)$ is an action introduced in step 1 $\big\}$ iff $X \neq \emptyset$.

It is easy to see that a PIIS $\mathcal{S}$ and its translation $\mathcal{S}'$ are equivalent in that every concrete system $\mathcal{S}(\overline{n})$ can simulate $\mathcal{S}'(\overline{n})$ and vice versa. $\mathcal{S}'$ is bigger than $\mathcal{S}$ by a polynomial factor in the number of transitions of the environment template.

We now introduce a notion of *agent-environment simulation* between the agent templates and the environment template. Intuitively, the existence of an agent-environment simulation between the agent and environment templates restricts the environment to one of shared resources, where each resource is accessible by exactly one agent at a given time. Formally, there is an agent-environment simulation between agent template $\mathcal{T}_i$ and $\mathcal{E}$ if $\mathcal{E}$ can simulate $\mathcal{T}_i$ only by means of the template states in which an action shared with the environment is enabled (see Figure 6.1) [1].

Definition 6.1 (Agent-environment simulation)**.** *A relation* $\sim_{aes} \subseteq L_i \times L_{\mathcal{E}}$ *is an* agent-environment simulation *between* $\mathcal{T}_i$ *and* $\mathcal{E}$ *if* $\iota_i \sim_{aes} \iota_{\mathcal{E}}$ *and whenever* $l_i \sim_{aes} l_E$ *the following condition holds:*

[1]Recall that $\mathbb{SMR}$ systems do not support GS actions. However the definition of agent-environment simulation includes GS actions as the definition will be reused for the analysis of $\mathbb{SGS}$ systems.

if $l_i \dashrightarrow_{A_i*} l_i' \dashrightarrow_a l_i''$, for some $a \in AE_i \cup MR_i \cup GS$, then there is l_E' with $l_E \dashrightarrow_a l_E'$ and $l_i'' \sim_{aes} l_E'$.

We write $\mathcal{T}_i \leq_{aes} \mathcal{E}$ to denote that there is an agent-environment simulation between $\mathcal{T}_i$ and $\mathcal{E}$. We write $\mathcal{T} \leq_{aes} \mathcal{E}$ if $\mathcal{T}_i \leq_{aes} \mathcal{E}$ for all $\mathcal{T}_i \in \mathcal{T}$. Intuitively, if $\mathcal{T} \leq_{aes} \mathcal{E}$, then an agent can always take the lock on a resource by synchronising with the environment via an agent-environment or a multi-role action. As we show below, following this synchronisation, said agent is the only agent that may synchronise with the environment. The agent releases the resources whenever the environment performs a loop in which case the other agents can synchronise with the environment.

Definition 6.2 (Environment loop). *A subsequence $g^i a^i \ldots g^j$, $j > i > 0$, of a path $g^1 a^1 \ldots$ in $\mathcal{S}(\overline{n})$ is an* environment loop *if $ls_E(g^i) = ls_E(g^j)$.*

So, if $\mathcal{T} \leq_{aes} \mathcal{E}$, then the concrete environment conforms to a certain looping behaviour whenever it synchronises between different agents. As we show below, an environment loop occurs whenever the environment synchronises between two different agents in successive synchronisations. If g is a global state occurring in a path from the first synchronisation to the latter, then we say that g has the *environment loop condition*.

Definition 6.3 (Environment loop condition). *A global state $\pi(d)$ in a path π in $\mathcal{S}(\overline{n})$ has the* environment loop condition*, denoted $ELC(\pi(d))$, if there is a pair of agents (i, r) and (j, q) with $(i, r) \neq (j, q)$ such that the following hold:*

(i) $\pi(e) \rightarrow_X \pi(e+1) \rightarrow_{Y*} \pi(d)$, *where* $X = AE_i^r \cup MR_i^r$ *and* $Y = \bigcup_{(x,y) \in \mathcal{A}(\overline{n})} A_x^y$;

(ii) $\pi(d) \rightarrow_{X'} \pi(d+1)$, *where* $X' = AE_j^q \cup MR_j^q \cup GS$.

In other words, a global state g in a path π has the environment loop condition if: (i) there is an agent who lastly synchronised with the environment earlier in the path; (ii) there is a different agent who firstly synchronised with the environment later in the path (see Figure 6.2).

Example 6.1. *Consider the following path of the concrete Train-Gate-Controller with two priori-*

$$\begin{tikzcd}
\iota(\bar{n}) \arrow[r, dashed, no head] & g \arrow[r, "AE_i^r \cup MR_i^r"] \arrow[rr, dotted, no head, bend left, "\text{Environment Loop } ls_E(g) = ls_E(g')"] & {} \arrow[r, "\bigcup_{(x,y)\in\mathcal{A}(\bar{n})} A_x^y *"] & g' \arrow[r, "AE_j^q \cup MR_j^q \cup GS"] & {} \arrow[r, dashed, no head] & {} \\
 & & & ELC(g') & &
\end{tikzcd}$$

Figure 6.2: Looping behaviour of a concrete environment in a path.

tised trains and two normal trains (Figure 3.7).

$$(W^1, W^2, PG, L^1, L^2) \to_{p\_enter^1} (T^1, W^2, R, L^1, L^2) \to_{p\_exit^1} (A^1, W^2, PG, L^1, L^2) \to_{p\_enter^2}$$
$$(A^1, T^2, R, L^1, L^2) \to_{p\_exit^2} (A^1, A^2, PG, L^1, L^2) \to_{n\_lock} (L^1, L^2, NG, A^1, A^2)$$

The states (A^1, W^2, PG, L^1, L^2) and (A^1, A^2, PG, L^1, L^2) have the environment loop condition, whereas the other states do not.

We now show that the environment loop condition is a sufficient condition for the occurrence of an environment loop. Specifically, we show that whenever $ELC(g)$ holds, the environment's local state in g is equal to its initial local state. Thus, intuitively, whenever $ELC(g)$ holds, an agent releases the lock on a shared resource to a different agent.

Lemma 6.1. *Consider a PIIS $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V}) \in \mathbb{SMR}$ with $\mathcal{T} \leq_{aes} \mathcal{E}$ and a path π in $\mathcal{S}(\bar{n})$. If $ELC(\pi(d))$ for some $d > 1$, then $[d]\pi$ is an environment loop.*

Proof. Assume that $ELC(\pi(d))$ for some $d > 1$. This means that there is a pair of agents $(i, i'), (j, j')$ with $(i, i') \neq (j, j')$ such that agent (i, i') is the last agent to synchronise with the environment in the prefix $[d]\pi$ of π, and agent (j, j') is the first agent to synchronise with the environment in the suffix $\pi[d]$ of π. We show that $ls_E(\pi(1)) = ls_E(\pi(d))$ by induction on d.

Suppose $d = 2$. Consider the template action a from which the concrete action performed in the first syncrhonisation of (j, j') with the environemnt in $[2]\pi$ has been instantiated. We have that $tls_j^{j'}(\pi(1)) \dashrightarrow_{A_j *} tls_j^{j'}(\pi(d'-1)) \dashrightarrow_a tls_j^{j'}(\pi(d'))$ for some $d' \geq 2$. As $\mathcal{T}_j \leq_{aes} \mathcal{E}$, the latter gives $ls_E(\pi(1)) \dashrightarrow_a l'_E$ for a template state l'_E with $tls_j^{j'}(\pi(d')) \sim_{aes} l'_E$. Since the set $\{l \in L_E \colon a \in P_E(l)\}$ is singleton, it follows that $ls_E(\pi(1)) = ls_E(\pi(d'-1))$. Additionally, $ls_E(\pi(2)) = ls_E(\pi(d'-1))$, as the subsequence $\pi(2), \ldots, \pi(d'-1)$ of π does not contain agent-environment or multi-role actions. Therefore, $ls_E(\pi(1)) = ls_E(\pi(2))$. So, the claim is true of paths with two states. Suppose that the claim is true for all paths with at most $x-1$ states for $x \geq 3$. Let π be a path with x states. We show that $ls_E(\pi(1)) = ls_E(\pi(x))$ in terms of two cases.

Case 1: $[x]\pi$ does not contain any $AE_j^{j'}$ action or any $MR_j^{j'}$ action. In this case we can proceed with the same argument used in the base step and conclude that $ls_E(\pi(1)) = ls_E(\pi(x))$.

Case 2: $[d]\pi$ contains an $AE_j^{j'}$ action or an $MR_j^{j'}$ action. Let q be the greatest integer in $\{1, \ldots, x\}$ with $\pi(q-1, Act) \in AE_j^{j'} \cup MR_j^{j'}$. From $\mathcal{T}_j \leq_{aes} \mathcal{E}$ we obtain that there is template state l_E with $tls_j^{j'}(\pi(q)) \sim_{aes} l_E$. Also, $l_E = ls_E(\pi(q))$, since $\pi(q-1, Act)$ is enabled by the environment's protocol at exactly one template state. Therefore, $tls_j^{j'}(\pi(q)) \sim_{aes} ls_E(\pi(q))$. From the latter observation and the observation that $tls_j^{j'}(\pi(q)) \dashrightarrow_{A_j*} tls_j^{j'}(\pi(d'-1))$ for some $d' \geq x$, the base step's argument applies to conclude that $ls_E(\pi(q)) = ls_E(\pi(x))$. It is left to prove that $[q]\pi$ is an environment loop, thereby proving that $[x]\pi$ is an environment loop. To show this, note that there is an agent $(r, r') \neq (j, j')$ which performs either an agent-environment action or a multi-role action in the path sequence $\pi(q), \ldots, \pi(x)$. Therefore, $ELC(\pi(q))$. So, $ls_E(\pi(1)) = ls_E(\pi(q))$ holds by the inductive hypothesis. $\square$

The above lemma reports a key consequence of the agent-environment simulation assumption. This will be central to the behavioural equivalence results shown in the next section. Lemma 6.1 can be interpreted as stating that the environment implements a mutual exclusion controller governing the access to shared resources.

6.2.2 Model checking procedure for $\mathbb{SMR}$ systems

The model checking procedure SMR for $\mathbb{SMR}$ systems is defined by Algorithm 3. Given a PIIS $\mathcal{S} \in \mathbb{SMR}$ and an $\overline{m}$-indexed ACTL*K$\backslash X$ formula $\forall_{\overline{v}}\phi(\overline{v})$, the procedure first establishes whether or not $\mathcal{T} \leq_{aes} \mathcal{E}$. Upon a successful simulation test, the procedure calculates the cutoff for the given system and specification.

The cutoff function *cutoff_SMR* maps a PIIS $\mathcal{S}$ of k roles and a k-tuple $\overline{m}$ of natural numbers into a k-tuple $\overline{c}$ of natural numbers that corresponds to the cutoff for $\overline{m}$-indexed formulae.

Algorithm 3 Parameterised model checking procedure for $\mathbb{SMR}$ systems.

```
1: procedure SMR($\mathcal{S}$,$\forall_{\bar{v}}\phi(\bar{v})$)
2:     if $\mathcal{T} \leq_{aes} \mathcal{E}$ then
3:         $\bar{c} = cutoff\_SMR(\mathcal{S}, \bar{m})$;
4:         for all $\bar{x}$ such that $\bar{m} \leq \bar{x} \leq \bar{c}$ do
5:             if $\mathcal{S}(\bar{x}) \not\models \phi[trivial]$ then
6:                 return $false$;
7:             end if
8:         end for
9:         return $true$;
10:    end if
11: end procedure
```

Informally, to calculate a cutoff, we require that each agent in the cutoff system, that is referred to by the atomic propositions and epistemic modalities in the trivial instantiation of an $\overline{m}$-indexed formula, is able to make any transition that the agent can make in any bigger system. This will enable us to establish that the cutoff system $\overline{m}$-stuttering simulates every bigger system. By Theorem 3.1, we can then conclude that the satisfaction of a formula on the cutoff system implies the satisfaction of the formula on every bigger system.

Consider an agent (i, j) indexing an atomic proposition or an epistemic modality in the trivial instantiation of an $\overline{m}$-indexed formula, and a transition by means of a multi-role action of agent (i, j) that is guarded by agent (r, q). As discussed in section 3.2, the enabling of the action depends on whether or not the action is enabled by the protocols at the current local states of the agents and the environment. If the action is enabled, then the agent (i, j) may update its state upon performing the action, whereas the agent (r, q) remains in its current local state. Obviously, agent (r, q) may not be present in cutoff system, as the index q can be arbitrarily large. Still, we can simulate the transition in the cutoff system by insisting on the presence of an agent, say (r, q'), in the local state of agent (r, q). Then, the instantiation of the multi-role action that is shared between the agents (i, j) and (r, q') is enabled, and so the agent (i, j) is able to simulate the transition of the bigger system by performing said instantiation. To simulate any multi-role action performed by agent (i, j), we insist on the presence of an agent in every local state enabling a multi-role action. In other words, for each agent template, in addition to the number of agents that need to be simulated, we require that the cutoff system composes as many agents as the cardinality of the set of template states that enable a multi-role action guarded by the template. We call this set *the action dependency set of the template.*

Definition 6.4 (Action dependency set)**.** *Given a PIIS* $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V}) \in \mathbb{SMR}$*, the* action dependency set $D_i \subseteq L_i$ *of agent template* $\mathcal{T}_i$ *is a subset of* $\mathcal{T}_i$*'s states that is defined as follows:*

$$D_i = \{l \in L_i \colon \text{ there are } 1 \leq j \leq k, a \in MR_{j,i} \text{ such that } a \in P_i(l)\}\,.$$

So, each action dependency set D_i reflects the states of template $\mathcal{T}_i$ that guard at least one multi-role action. The cutoff function is defined in terms of $\overline{m}$ and the action dependency sets for the agent templates.

Definition 6.5 (Cutoff function for $\mathbb{SMR}$ systems)**.** *The* cutoff function $cutoff\_SMR$ *is defined for* $\mathbb{SMR}$ *systems as follows:*

$$cutoff\_SMR(\mathcal{S}, \overline{m}) = (\max(1, \overline{m}(1) + |D_1|), \ldots, \max(1, \overline{m}(k) + |D_k|))\,,$$

for any $\mathcal{S} \in \mathbb{SMR}$ *with* $k \geq 1$ *roles and any* $\overline{m} \in \mathbb{N}^k$*.*

Following the cutoff calculation, SMR checks the set $\{\mathcal{S}(\overline{x}) \colon \overline{m} \leq \overline{x} \leq \overline{c}\}$ of concrete systems against the trivial instantiation $\phi[trivial]$ of $\forall_{\overline{v}}\phi(\overline{v})$. If $\phi[trivial]$ is not satisfied by at least one system, then the procedure returns false, otherwise it returns true. We assess the soundness of the SMR procedure in section 6.2.4. First, we exemplify it by means of the robot foraging scenario.

6.2.3 Verifying the robot foraging scenario

In section 3.2.1 we represented the robot foraging scenario as a PIIS $\mathcal{S}_{RFS}$ composed of an agent template TR encoding the robots and an agent template TFS encoding the food sources. For the encoding we used the multi-role actions $observe$, $reached$, $deposit$, and $scan$. Observe that $observe$ is guarded by TFS, $reached$ is guarded by TR, $deposit$ is guarded by TFS, and $scan$ is guarded by TFS; thus, $\mathcal{S}_{RFS} \in \mathbb{SMR}$. In section 3.3 we expressed the property "whenever a food source is found, every robot knows that the source is found" in the following $(1, 1)$-indexed formula:

$$\phi_{RFS} = \forall_{(\{u\},\{x\})} AG\left((f, x) \rightarrow K^{u}_{TR}(f, x)\right),$$

where u is variable of TR, x is a variable of TFS and the atomic proposition f holds in the template state in which the template food source is "found".

We now show how $\mathcal{S}_{RFS}$ can be verified using the SMR procedure. Clearly, $\mathcal{S}_{RFS}$ satisfies the agent-environment simulation assumption as the system does not specify an environment template. Therefore, we may proceed to compute the cutoff function $\mathit{cutoff\_SMR}(\mathcal{S}_{RFS}, (1, 1))$. To do this, we first calculate the action dependency sets. We have that $D_{TR} = \{MF\}$ and $D_{TFS} = \{N\_F, F\}$ for TR and TFS, respectively. Hence,

$$\bar{c} = \mathit{cutoff\_SMR}(\mathcal{S}_{RFS}, (1, 1)) = (1 + 1, 1 + 2) = (2, 3)$$

Thus, we need to check whether or not $\mathcal{S}_{RFS}(\overline{x}) \models \phi_{RFS}[\mathit{trivial}]$, for $(1, 1) \leq \overline{x} \leq (2, 3)$. These checks can be performed on a standard model checker; the result is true thereby establishing the correctness of the protocol irrespectively of the number of robots and the number of food sources.

6.2.4 Proof of soundness

Theorem 6.1. *Let $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V}) \in \mathbb{SMR}$ be a PIIS and $\forall_{\overline{v}}\phi(\overline{v})$ be an $\overline{m}$-indexed formula. If $\mathcal{T} \leq_{aes} \mathcal{E}$, then SMR$(\mathcal{S}, \forall_{\overline{v}}\phi(\overline{v}))$ returns true iff $\mathcal{S} \models \forall_{\overline{v}}\phi(\overline{v})$.*

To prove this result we need to establish some intermediate steps. Firstly observe that by the definition of the PMCP and Theorem 3.1, it suffices to show that if $\mathcal{T} \leq_{aes} \mathcal{E}$, then: (i) the cutoff system $\mathcal{S}(\bar{c})$ $\overline{m}$-stuttering simulates every bigger system; (ii) every bigger system $\overline{m}$-stuttering simulates $\mathcal{S}(\bar{c})$. As related states in each of the simulations need only to agree on the atomic propositions indexed by the agents in $\mathcal{A}(\overline{m})$, we only have to simulate transitions taken by the agents in $\mathcal{A}(\overline{m})$. We first show the former.

Part A: The cutoff system $\overline{m}$-stuttering simulates every bigger system

Consider an arbitrarily big system $\mathcal{S}(\overline{n})$ with $\overline{n} \geq \bar{c}$, where $\mathcal{S}(\bar{c})$ is the cutoff system. To simulate $\mathcal{S}(\overline{n})$, $\mathcal{S}(\bar{c})$ first executes the *action dependency path*. The path results in an agent to be in every local state guarding a multi-role action. Following the execution of the action dependency path, $\mathcal{S}(\overline{c})$ can simulate the multi-role transitions of $\mathcal{S}(\overline{n})$ in the way described earlier. More specifically, the action dependency path associates the concrete agents in $\mathcal{A}(\overline{c}) \setminus \mathcal{A}(\overline{m})$ with the action dependency sets as follows. Let $\lambda_i : D_i \to \{(i, \overline{m}(i)+1), \ldots, (i, \overline{c}(i))\}$ be a bijective mapping from the set D_i of template states to the set $\{(i, \overline{m}(i)+1), \ldots, (i, \overline{c}(i))\}$ of concrete agents. Given $l \in D_i$, $\lambda_i(l)$ denotes the concrete agent of role i which moves to template state l via the execution of the action dependency path. Then, each agent $(i,j) \in \mathcal{A}(\overline{c}) \setminus \mathcal{A}(\overline{m})$ remains forever in its corresponding state $\lambda_i^{-1}((i,j))$. Thus, $\mathcal{S}(\overline{c})$ can mimic the multi-role transitions of the agents in $\mathcal{A}(\overline{m})$ in $\mathcal{S}(\overline{n})$ as follows: for each multi-role action $(b, (i,r), (\mathbf{j}, \mathbf{q}))$ shared by the agents $(i,r) \in \mathcal{A}(\overline{m}), (j,q) \in \mathcal{A}(\overline{n})$, guarded by (j,q), and performed at a global state g in $\mathcal{S}(\overline{n})$, the multi-role action $(b, (i,r), (\mathbf{j}, \mathbf{q}'))$ shared by the agents (i,r), (j,q'), where $q' = \lambda_j \left(tls_j^q(g) \right)$, and guarded by (j,q') is performed in $\mathcal{S}(\overline{c})$.

We now give a formal definition of the action dependency path. The path is inductively defined so that at each step an agent $(i,j) \in \mathcal{A}(\overline{c}) \setminus \mathcal{A}(\overline{m})$ moves to its associated state $\lambda_i^{-1}((i,j))$. Following Lemma 6.1, however, the definition insists on the occurrence of an environment loop at each step. If an agent cannot move to its associated state while the environment performs an environment loop, then said agent remains in its initial state. This is because the environment is locked on synchronising with exactly one agent unless an environment loop occurs.

Definition 6.6 (Action dependency path)**.** *Let $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V}) \in \mathbb{SMR}$ be a PIIS of k roles. Consider $D = D_1 \cup \ldots \cup D_k$ to be the union of the action-dependency sets. For a global state g, assume $D(g) = \{l \in D\colon$ there is an agent (i,r) with $tls_i^r(g) = l\}$ to be the set of template states in D that appear in g. Then, for $\overline{m} \in \mathbb{N}^k$ and $\overline{c} = cutoff\_SMR(\mathcal{S}, \overline{m})$, the* action dependency path $\mathfrak{p}$ *in $\mathcal{S}(\overline{c})$ is inductively defined as follows.*

- $\mathfrak{p}^0 = \iota(\overline{c})$.
- *Let $X \subseteq \Pi(\mathfrak{p}^i[])$ be a set of finite paths such that $\pi \in X$ if and only if π satisfies the following conditions:*

  i. π is an environment loop;

ii. $D(\pi[]) = D(\mathfrak{p}^i[]) \cup \{l\}$ *for some* $l \in D \setminus D(\mathfrak{p}^i[])$;

iii. *Every action occurring in* π *is in* $A_j^q \cup AE_j^q \cup MR_j^q$, *where* (j,q) *is the agent associated with* l.

If $X = \emptyset$, *then* $\mathfrak{p}^{i+1} = \mathfrak{p}^i$. *Otherwise,* $\mathfrak{p}^{i+1} = \mathfrak{p}^i \circ \pi$ *for an arbitrary* $\pi \in X$.

Example 6.2. *Consider the robot foraging scenario discussed in Section 6.2.3, where the action dependency sets were calculated to be* $D_{TR} = \{MF\}$ *for the template robot, and* $D_{TFS} = \{N\_F, F\}$ *for the template food source. The cutoff was shown to be* $\bar{c} = (2,3)$ *for* $(1,1)$*-indexed formulae. We now construct the action dependency path. To do this, we first associate the concrete agents with the action dependency sets by the following mappings:*

$$\lambda_{TR} = \{(TR, 2) \to MF\}$$
$$\lambda_{TFS} = \{(TFS, 2) \to N\_F, (TFS, 3) \to F\}.$$

So, the concrete robot 2 is associated with the state MF, *the concrete food source 2 is associated with the state* N_F, *and the concrete food source 3 is associated with the state* F. *For the base step, we have that* $\mathfrak{p}^0 = \iota(\bar{c})$, *where* $\iota(\bar{c}) = ((R,1),(R,2),(N\_F,1),(N\_F,2),(N\_F,3))$ *is the initial global state composed of the local states* $(R,1),(R,2)$ *of concrete robots 1 and 2, and the local states* $(N\_F,1),(N\_F,2),(N\_F,3)$ *of concrete food sources 1,2, and 3. Clearly, the agent* $(TFS, 2)$ *is already in its associated state* N_F *in* $\iota(\bar{c})$. *Now consider the path*

$$\begin{aligned}\pi =& ((R,1),(R,2),(N\_F,1),(N\_F,2),(N\_F,3)) \to_{(search,2)} \\ & ((R,1),(RW,2),(N\_F,1),(N\_F,2),(N\_F,3)) \to_{(observe,(TR,2),(\mathbf{TFS},\mathbf{2}))} \\ & ((R,1),(MF,2),(N\_F,1),(N\_F,2),(N\_F,3)).\end{aligned}$$

We have that π *is an environment loop. Also,* $D(\pi[]) = D(\mathfrak{p}^0[]) \cup \{MF\}$, *where* $MF \in D \setminus D(\mathfrak{p}^0[])$. *Additionally, every action in* π *is in* $A_{TR}^2 \cup AE_{TR}^2 \cup MR_{TR}^2$. *Thus,* π *satisfies all three conditions of Definition 6.6. So,* $\mathfrak{p}^1 = \pi$. *For the next step, consider the path*

$$\begin{aligned}\pi' = & ((R,1),(MF,2),(N\_F,1),(N\_F,2),(N\_F,3) \to_{(reached,(TFS,3),(\mathbf{TR},\mathbf{2}))} \\ & ((R,1),(MF,2),(N\_F,1),(N\_F,2),(F,3))\end{aligned}$$

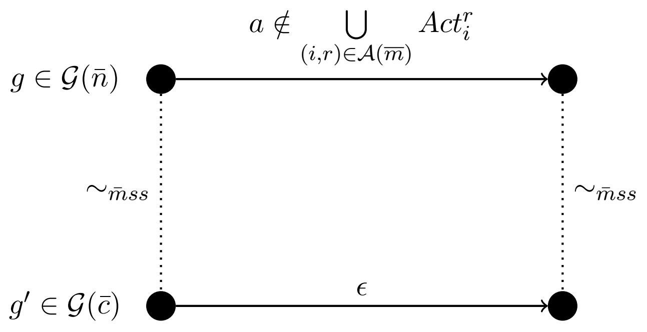
Obviously, π' is an environment loop. Also, $D(\pi'[]) = D(\mathfrak{p}^1[]) \cup \{F\}$, where $F \in D \setminus D(\mathfrak{p}^1[])$. Additionally, every action in π' is in $A^3_{TFS} \cup AE^3_{TR} \cup MR^3_{TR}$. Therefore, $\mathfrak{p}^2$ is equal to the following:

$$\begin{aligned}\mathfrak{p}^2 =&((R,1),(R,2),(N\_F,1),(N\_F,2),(N\_F,3)) \rightarrow_{(search,2)}\\ &((R,1),(RW,2),(N\_F,1),(N\_F,2),(N\_F,3)) \rightarrow_{(observe,(TR,2),(\mathbf{TFS},\mathbf{2}))}\\ &((R,1),(MF,2),(N\_F,1),(N\_F,2),(N\_F,3)) \rightarrow_{(reached,(TFS,3),(\mathbf{TR},\mathbf{2}))}\\ &((R,1),(MF,2),(N\_F,1),(N\_F,2),(F,3)).\end{aligned}$$

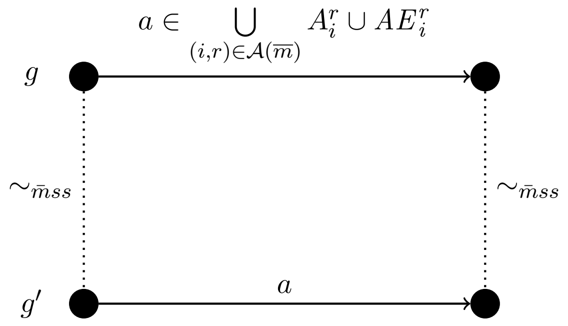
As $D(\mathfrak{p}^2) = D$, $\mathfrak{p}^2$ is the required action dependency path.

In other words, every template state in D that can be reached by an agent via an environment loop is a local state for an agent in $\mathfrak{p}[]$. Moreover, there is no global state g in any bigger system satisfying the environment loop condition and also satisfying $D(g) \setminus D(\mathfrak{p}[]) \neq \emptyset$. So, $\mathcal{S}(\overline{c})$ can simulate the multi-role transitions of the agents in $\mathcal{A}(\overline{m})$ in $\mathcal{S}(\overline{n})$ with the execution of the action dependency path. Other types of actions are simulated as follows. For an agent-environment action or an asynchronous action performed by the agents in $\mathcal{A}(\overline{m})$ in $\mathcal{S}(\overline{n})$, $\mathcal{S}(\overline{c})$ performs the same action. Finally, for any type of action performed by an agent not in $\mathcal{A}(\overline{m})$ in $\mathcal{S}(\overline{n})$, $\mathcal{S}(\overline{c})$ performs the null action. This implies that the state of the environment in $\mathcal{S}(\overline{n})$ may change, whereas the state of the environment in $\mathcal{S}(\overline{c})$ remains the same. However, the environment in $\mathcal{S}(\overline{c})$ is not blocked from synchronising with an agent in $\mathcal{A}(\overline{m})$ at a later point. For example, assume the path $g^1a^1g^2 \dots a^{x-1}g^x$ in $\mathcal{S}(\overline{n})$, where $a^1, a^{x-1} \in \bigcup_{(i,j)\in\mathcal{A}(\overline{m})} AE^j_i$ and $a^2, \dots, a^{x-2} \notin \bigcup_{(i,j)\in\mathcal{A}(\overline{m})} A^j_i \cup AE^j_i \cup MR^j_i$. Clearly, the state of the environment in $\mathcal{S}(\overline{n})$ may be updated after the environment synchronises on the $a^2, \dots, a^{x-2}$ actions. Even so, the environment in $\mathcal{S}(\overline{c})$ may still synchronise on the a^{x-1} action after performing the null actions. This is because both g^2 and g^{x-1} satisfy the environment loop condition. Therefore, the state of the environment in g^2 is the same to its state in g^{x-1}. Intuitively, whenever an agent not in $\mathcal{A}(\overline{m})$ in $\mathcal{S}(\overline{n})$ takes the lock on a resource via synchronising with the environment, it has to release the resource, i.e., an environment loop occurs, before another agent in $\mathcal{A}(\overline{m})$ can take the lock on a resource.

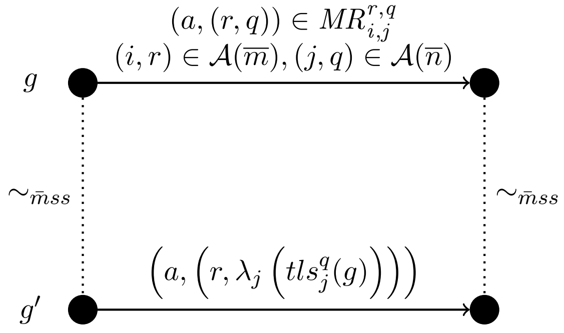
Lemma 6.2. *Let $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V}) \in \mathbb{SMR}$ be a PIIS of k roles with $\mathcal{T} \leq_{aes} \mathcal{E}$. Let $\overline{m} \in \mathbb{N}^k$. Then, $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{c})$ for all $\overline{n} \geq \overline{c}$, where $\overline{c} = cutoff\_SMR(\mathcal{S}, \overline{m})$.*


$a \notin \bigcup_{(i,r)\in\mathcal{A}(\overline{m})} Act_i^r$
$g \in \mathcal{G}(\bar{n})$
$\sim_{\bar{m}ss}$
$\sim_{\bar{m}ss}$
ϵ
$g' \in \mathcal{G}(\bar{c})$

(a) Simulation of an action performed by an agent not in $\mathcal{A}(\overline{m})$.


$a \in \bigcup_{(i,r)\in\mathcal{A}(\overline{m})} A_i^r \cup AE_i^r$
g
$\sim_{\bar{m}ss}$
$\sim_{\bar{m}ss}$
a
g'

(b) Simulation of an asynchronous action or an agent-environment action performed by an agent in $\mathcal{A}(\overline{m})$.


$(a,(r,q)) \in MR_{i,j}^{r,q}$
$(i,r) \in \mathcal{A}(\overline{m}), (j,q) \in \mathcal{A}(\overline{n})$
g
$\sim_{\bar{m}ss}$
$\sim_{\bar{m}ss}$
$\left(a, \left(r, \lambda_j \left(tls_j^q(g)\right)\right)\right)$
g'

(c) Simulation of a multi-role action performed by an agent in $\mathcal{A}(\overline{m})$.

Figure 6.3: The $\overline{m}$-stuttering simulation of a bigger system $\mathcal{S}(\overline{n})$ by the cutoff system $\mathcal{S}(\overline{c})$.

Proof. Let $\overline{n} \geq \overline{c}$ be arbitrary. We show that $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{c})$. The simulation relation we define follows Figure 6.3. It ensures that whenever an action is performed by an agent in $\mathcal{S}(\overline{n})$ that is required to satisfy condition (i) of $\overline{m}$-stuttering simulation, the agent can also perform the action in $\mathcal{S}(\overline{c})$. In line with this, the state of the environment is defined to allow the actions to be performed. The simulation relation $\sim_{\overline{m}ss} = (R_1 \cap R_2 \cap R_3) \cup R_4 \subseteq \mathcal{G}(\overline{n}) \times \mathcal{G}(\overline{c})$ is defined as follows.

$$(g, g') \in R_1 \text{ iff } ls_i^r(g) = ls_i^r(g'), \text{ for } (i, r) \in \mathcal{A}(\overline{m})$$

The above ensures that the local states of the agents in $\mathcal{A}(\overline{m})$ are the same in related global states of $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{c})$. The following ensures that the agents in $\mathcal{A}(\overline{c}) \setminus \mathcal{A}(\overline{m})$ in $\mathcal{S}(\overline{c})$ are in their associated local states as per the action dependency path.

$$(g, g') \in R_2 \text{ iff } ls_i^r(\mathfrak{p}[]) = ls_i^r(g'), \text{ for } (i, r) \notin \mathcal{A}(\overline{m})$$

$(R_1 \cap R_2)$-related states must also agree on the state of the environment. Assume $(g, g') \in \mathcal{G}(\overline{n}) \times \mathcal{G}(\overline{c})$. Suppose that there is a state g^1 reachable from g at which an agent-environment action or a multi-role action of the agents in $\mathcal{A}(\overline{m})$ is enabled. Clearly, for the action to be enabled, the environment is in a local state at which the environment's protocol enables the action. Further assume that g^1 can be reached by asynchronous actions and by actions of agents not in $\mathcal{A}(\overline{m})$. As previously discussed, $\mathcal{S}(\overline{c})$ simulates these actions by performing asynchronous and null actions. These do not change the state of the environment in $\mathcal{S}(\overline{c})$. So, we define R_3 to insist on the equality of the environment's states in g' and g^1.

$$(g, g') \in R_3 \text{ iff } g \rightarrow_{X*} g^1 \rightarrow_a g^2 \text{ implies that } ls_E(g') = ls_E(g^1), \text{ where}$$
$$a \in \bigcup_{(i,r)\in\mathcal{A}(\overline{m})} AE_i^r \cup MR_i^r \text{ and } X = \bigcup_{(i,r)\in\mathcal{A}(\overline{m})} A_i^r \cup \bigcup_{(i,r)\notin\mathcal{A}(\overline{m})} Act_i^r$$

Finally, to simulate the multi-role actions in $\mathcal{S}(\overline{n})$, $\mathcal{S}(\overline{c})$ first executes the action dependency path. R_4 suggests exactly this.

$$(\iota(\overline{n}), g) \in R_4 \text{ iff } g \text{ appears in } \mathfrak{p}$$

We show that $\sim_{\overline{m}ss}$ is an $\overline{m}$-stuttering simulation between $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{c})$. As $\iota(\overline{c}) = \mathfrak{p}(1)$, we have that $(\iota(\overline{n}), \iota(\overline{c})) \in R_4$, therefore $\iota(\overline{n}) \sim_{\overline{m}ss} \iota(\overline{c})$. Now assume that $g \sim_{\overline{m}ss} g'$ for an arbitrary pair of global states in $\mathcal{G}(\overline{n}) \times \mathcal{G}(\overline{c})$. We show the simulation requirements (i) and (iii) of $\overline{m}$-stuttering simulation (see Definition 3.15; from Theorem 3.1, we only have to show simulation requirements (i) and (iii)). We have that $(g, g') \in R_1 \cap R_2 \cap R_3$ or $(g, g') \in R_4$. The first simulation requirement follows trivially by the definition of R_1 and by observing that each agent in $\mathcal{A}(\overline{m})$ remains in its initial local state in $\mathfrak{p}$.

To show the simulation requirement (iii), let $\pi \in \Pi(g)$. We inductively construct a path $\pi' \in \Pi(g')$ as required by the $\overline{m}$-stuttering-simulation.

For the base step, we have two cases:

- $(g, g') \in R_4$. Then, there is a $q \geq 1$ such that $g' = \mathfrak{p}(q)$. Consider $\pi' = \mathfrak{p}[q]$. Define the first blocks B_1, B'_1 by $B_1 = g$ and $B'_1 = \mathfrak{p}(q), \mathfrak{p}(q+1), \ldots, \mathfrak{p}[]$. B_1, B'_1 are as required: for each x in B'_1, $g \sim_{\overline{m}ss} x$, since $(g, x) \in R_4$.
- $(g, g') \notin R_4$. In this case, define $\pi' = g'$, $B_1 = g$, and $B'_1 = g'$.

For the inductive step, assume that we have already constructed a prefix $[x]\pi$, a prefix $[x']\pi'$, and a partition of the states in $[x]\pi$ and $[x']\pi'$ into corresponding blocks. We now define $[x'+1]\pi'$ and the next blocks B_{x+1} and B'_{x+1} by considering cases on the type of the action $\pi(x, Act)$.

- if $\pi(x, Act) \in A^r_i \cup AE^r_i \cup MR^{r,q}_{i,j}$ for $(i, r) \in \mathcal{A}(\overline{m})$, $(j, q) \in \mathcal{A}(\overline{m})$, then $\pi'(x', Act) = \pi(x, Act)$.
- if $\pi(x, Act) = (a, (i, r), (\mathbf{j}, \mathbf{q})) \in MR^{r,q}_{i,j}$ for $(i, r) \in \mathcal{A}(\overline{m})$, $(j, q) \in \mathcal{A}(\overline{n}) \setminus \mathcal{A}(\overline{m})$, then $\pi'(x', Act) = (a, (i, r), ((\mathbf{j}, \mathbf{q}'))$, where $q' = \lambda_j(tls^q_j(\pi(x)))$;
- Otherwise, $\pi'(x', Act) = \epsilon$.

Let $B_{x+1} = \pi(x+1)$ and $B'_{x+1} = \pi'(x'+1)$. We have to show that $\pi'(x') \rightarrow_{\pi'(x', Act)} \pi'(x'+1)$ is a valid transition. The case follows trivially when $\pi'(x', Act)$ is an asynchronous action or the joint null action. If $\pi'(x', Act)$ is an agent-environment action, then we have to show that the state of the environment enables the action in $\pi'(x')$. If $\pi'(x', Act)$ is a multi-role action, say $\pi'(x', Act) = (a, (i, r), ((\mathbf{j}, \mathbf{q}')) \in MR_{i,j}^{r,q'}$, then we additionally have to show that the state of the agent (j, q') enables the action as well. So, it suffices to show the case for multi-role actions. That is, we need to show: (i) $a \in P_{\mathcal{E}}(l_E)$, where $l_E = ls_E(\pi'(x'))$; (ii) $a \in P_j(l)$, where $l = tls_j^{q'}(\pi'(x'))$. We show (i) by means of two cases.

1. There is a y with $1 \leq y \leq x - 1$ such that $\pi(y, Act) \in \bigcup_{(i,r)\in\mathcal{A}(\overline{m})} AE_i^r \cup MR_i^r$. Let z be the greatest integer with $1 \leq z \leq x - 1$ that satisfies $\pi(z, Act) \in \bigcup_{(i,r)\in\mathcal{A}(\overline{m})} AE_i^r \cup MR_i^r$. By construction of π', there is a greatest z' with $1 \leq z' \leq x' - 1$ that satisfies $\pi'(z', Act) \in \bigcup_{(i,r)\in\mathcal{A}(\overline{m})} AE_i^r \cup MR_i^r$. Since $\pi(z, Act) = \pi'(z', Act)$, it follows that $ls_E(\pi(z + 1)) = ls_E(\pi'(z' + 1))$. By construction of π', the path $\pi'(z' + 1), \ldots, \pi'(x')$ does not contain agent-environment actions or multi-role actions. Hence, $ls_E(\pi'(z' + 1)) = ls_E(\pi'(x'))$. The latter gives $ls_E(\pi(z + 1)) = ls_E(\pi'(x'))$.

   We now show that $ls_E(\pi(z + 1)) = ls_E(\pi(x))$. If the path $\pi(z + 1), \ldots, \pi(x)$ does not contain an agent-environment action or a multi-role action, then the environment does not change its state in $\pi(z + 1) \ldots \pi(x)$; therefore, the claim follows trivially. Otherwise, if there is at least one agent-environment action or at least one multi-role action in $\pi(z + 1), \ldots, \pi(x)$, then $ELC(\pi(z + 1))$ and $ELC(\pi(x))$. Therefore, Lemma 6.1 gives that $[z + 1]\pi$ and $[x]\pi$ are environment loops. It follows that $ls_E(\pi(z + 1)) = ls_E(\pi(x))$.

   From the above two observations we deduce $ls_E(\pi(x)) = ls_E(\pi'(x'))$. As $a \in P_{\mathcal{E}}(ls_E(\pi(x)))$, we obtain $a \in P_{\mathcal{E}}(l_E)$.

2. For all y with $1 \leq y \leq x - 1$ we have that $\pi(y, Act) \notin \bigcup_{(i,r)\in\mathcal{A}(\overline{m})} AE_i^r \cup MR_i^r$. There are two cases.

   (a) $(g, g') \in R_3$. As $g \rightarrow_{X*} \pi(x) \rightarrow_a \pi(x + 1)$, we obtain $ls_E(g') = ls_E(\pi(x))$. By construction of π', $ls_E(g') = ls_E(\pi'(x'))$, therefore $ls_E(\pi(x)) = ls_E(\pi'(x'))$, and therefore $a \in P_{\mathcal{E}}(l_E)$.

   (b) $(g, g') \notin R_3$. Then, it must be the case that $(g, g') \in R_4$. So, $ls_E(g) = \iota_E$. By construction of π', $ls_E(\pi'(x')) = \iota_E$. We show that $ls_E(\pi(x)) = \iota_E$ is also the case. If there is no agent-environment action or multi-role action in $[x]\pi$, then the case follows trivially. Otherwise, we have $ELC(\pi(x))$, hence $[x]\pi$ is an environment loop. Therefore, $ls_E(\pi(x)) = \iota_E$. Thus, $ls_E(\pi(x)) = ls_E(\pi'(x'))$. Hence, $a \in P_{\mathcal{E}}(l_E)$.

To show (ii), let ρ in $\mathcal{S}(\overline{n})$ be a finite path of length len such that $\rho(len) = g$, and consider the concatenation $\gamma = \rho \circ \pi$ of ρ and π. If there is a y with $1 \leq y \leq x + len$ and $ELC(\gamma(y))$, then let z be the greatest integer with $ELC(\gamma(z))$; otherwise, let $z = 1$. As there is no state in $\gamma(z+1), \ldots, \gamma(len + x)$ that satisfies the environment loop condition, and as $\gamma(len + x, Act) = (a, (i,r), ((\mathbf{j}, \mathbf{q})) \in MR_{i,j}^{r,q}$ for some $(j,q) \in \mathcal{A}(\overline{n}) \setminus \mathcal{A}(\overline{m})$, it follows that only agent (i,r) (possibly) synchronises with the environment in the path $\gamma(z), \ldots, \gamma(len + x)$. By construction of $\mathfrak{p}$, $D(\gamma(z)) \subseteq D(\mathfrak{p}[])$. Therefore, $(D(\gamma(len + x)) \setminus \{tls_i^r(\gamma(len + x))\}) \subseteq D(\mathfrak{p}[])$. That is, $(D(\pi(x)) \setminus \{tls_i^r(\pi(x))\}) \subseteq D(\mathfrak{p}[]) \subseteq D(\pi'(x'))$. This means that the template state of agent (j,q) in $\pi(x)$ that enables the action a is in $D(\pi(x'))$. Therefore, by construction of $\mathfrak{p}$, $ls_j^q(\pi(x)) = ls_j^{q'}(\pi'(x'))$. It follows that $a \in P_j(l)$.

We have thus shown that $\pi'(x') \rightarrow_{\pi'(x',Act)} \pi'(x'+1)$ is a valid transition. It is left to show that $\pi(x+1) \sim_{\overline{m}ss} \pi'(x'+1)$. To do this, we show that $(\pi(x+1), \pi'(x'+1)) \in R_1 \cap R_2 \cap R_3$. The case of $(\pi(x+1), \pi'(x'+1)) \in R_1 \cap R_2$ follows trivially. Consider the case of $(\pi(x+1), \pi'(x'+1)) \in R_3$. Suppose that $\pi(x+1) \rightarrow_{X*} \pi(x+d) \rightarrow_a \pi(x+d+1)$, for some $d \geq 1$, $a \in \bigcup_{(i,r)\in\mathcal{A}(\overline{m})} AE_i^r \cup MR_i^r$, and $X = \bigcup_{(i,r)\in\mathcal{A}(\overline{m})} A_i^r \cup \bigcup_{(i,r)\notin\mathcal{A}(\overline{m})} Act_i^r$. We need to show that $ls_E(\pi'(x'+1)) = ls_E(\pi(x+d))$. If the path $\pi(x+1) \ldots \pi(x+d)$ consists only of asynchronous actions, then the case follows trivially. If not, then we obtain $ELC(\pi(x+1))$ and $ELC(\pi(x+d))$. Therefore, $ls_E(\pi'(x'+1)) = ls_E(\pi(x+d))$. Consequently, $(\pi(x+1), \pi'(x'+1)) \in R_3$.

It follows that simulation requirement (iii) is satisfied. As (g, g') was arbitrary, $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{c})$. $\square$

Lemma 6.2 concludes part A of the proof for Theorem 6.1: the cutoff system $\overline{m}$-stuttering simulates every bigger system. The following constitutes part B of the proof: every bigger system $\overline{m}$-stuttering simulates the cutoff system.

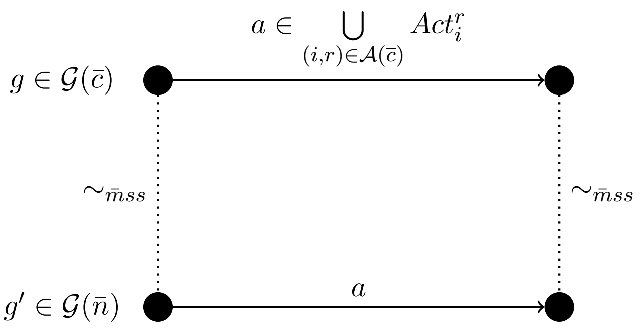

$a \in \bigcup_{(i,r)\in\mathcal{A}(\bar{c})} Act_i^r$
$g \in \mathcal{G}(\bar{c})$
$\sim_{\overline{m}ss}$
$\sim_{\overline{m}ss}$
$g' \in \mathcal{G}(\bar{n})$
a

Figure 6.4: The $\overline{m}$-stuttering simulation of the cutoff system $\mathcal{S}(\overline{c})$ by a bigger system $\mathcal{S}(\overline{n})$.

Part B: Every bigger system $\overline{m}$-stuttering simulates the cutoff system

Figure 6.4 shows the $\overline{m}$-stuttering simulation between $\mathcal{S}(\overline{c})$ and $\mathcal{S}(\overline{n})$. $\mathcal{S}(\overline{n})$ performs precisely every action performed by $\mathcal{S}(\overline{c})$ while it lets the agents not in $\mathcal{A}(\overline{c})$ stutter at their initial states.

Lemma 6.3. *Let $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V}) \in \mathbb{SMR}$ be a PIIS of k roles with $\mathcal{T} \leq_{aes} \mathcal{E}$. Let $\overline{m} \in \mathbb{N}^k$. Then, $\mathcal{S}(\overline{c}) \leq_{\overline{m}ss} \mathcal{S}(\overline{n})$ for all $\overline{n} \geq \overline{c}$, where $\overline{c} = cutoff\_SMR(\mathcal{S}, \overline{m})$.*

Proof. Choose an arbitrary $\overline{n} \geq \overline{c}$. We show that $\mathcal{S}(\overline{c}) \leq_{\overline{m}ss} \mathcal{S}(\overline{n})$. We do this by letting each additional agent in $\mathcal{S}(\overline{n})$ stutter at its initial state. Then, $\mathcal{S}(\overline{n})$ can simulate $\mathcal{S}(\overline{c})$ by performing the same actions performed in $\mathcal{S}(\overline{c})$.

Define $R_1, R_2, R_3 \subseteq \mathcal{G}(\overline{c}) \times \mathcal{G}(\overline{n})$ as follows.

$$(g, g') \in R_1 \text{ iff } ls_i^r(g') = ls_i^r(g), \text{ for } (i,r) \in \mathcal{A}(\overline{c})$$
$$(g, g') \in R_2 \text{ iff } tls_i^r(g') = \iota, \text{ for } (i,r) \notin \mathcal{A}(\overline{c})$$
$$(g, g') \in R_3 \text{ iff } ls_E(g) = ls_E(g')$$

Let $\sim_{\overline{m}ss} = R_1 \cap R_2 \cap R_3$; the information encoded in a state in $\mathcal{S}(\overline{c})$ is present in a related state in $\mathcal{S}(\overline{n})$ by R_1; the additional agents in $\mathcal{S}(\overline{n})$ are left stuttering at their initial states by R_2; the environment's states are equal in every pair of related states by R_3.

We show that $\sim_{\overline{m}ss}$ is an $\overline{m}$-stuttering simulation between $\mathcal{S}(\overline{c})$ and $\mathcal{S}(\overline{n})$. It should be clear that $\iota(\overline{c}) \sim_{\overline{m}ss} \iota(\overline{n})$. To show the simulation requirements (i) and (iii), assume that $g \sim_{\overline{m}ss} g'$ for an arbitrary pair of global states in $\mathcal{G}(\overline{c}) \times \mathcal{G}(\overline{n})$. Requirement (i) follows by definition of R_1. To show the requirement (iii), let $\pi \in \Pi(g)$. Define a path π' by $g'\pi(1, Act), \pi(2, Act), \ldots$. Clearly,

$\pi' \in \Pi(g')$ and $\pi(j) \sim_{\overline{m}ss} \pi'(j)$, for all $j \geq 1$. Thus, define $B_1, B_2, \ldots$ and $B'_1, B'_2, \ldots$ to be a partition of π and π', respectively, into singleton blocks. It follows that $\mathcal{S}(\overline{c}) \leq_{\overline{m}ss} \mathcal{S}(\overline{n})$. □

Corollary 6.1. *Let $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V}) \in \mathbb{SMR}$ be a PIIS with $\mathcal{T} \leq_{aes} \mathcal{E}$. Let $\forall_{\overline{v}}\phi(\overline{v})$ be an $\overline{m}$-indexed formula. Then, $\mathcal{S}(\overline{c}) \models \forall_{\overline{v}}\phi(\overline{v})$ iff $\forall \overline{n} \geq \overline{c}. \mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v})$, where $\overline{c} = cutofff\_SMR(S, \overline{m})$.*

Proof. Let $\overline{n} \geq \overline{c}$ be arbitrary. By Theorem 3.1 it suffices to show that $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{c})$ and $\mathcal{S}(\overline{c}) \leq_{\overline{m}ss} \mathcal{S}(\overline{n})$. The former is given by Lemma 6.2, and the latter is given by Lemma 6.3. □

This concludes the proof of Theorem 6.1. Corollary 6.1 provides a constructive methodology for solving the PMCP by giving the conditions under which the problem can be solved by checking each concrete system up to the cutoff system.

6.3 Verifying $\mathbb{SGS}$ systems

We now investigate the $\mathbb{SGS}$ class of systems. The class is generated from agent templates defined on asynchronous, agent-environment and global-synchronous actions. We show that the existence of an agent-environment simulation between the agent and environment templates guarantees a cutoff. We show how to calculate the cutoff under these conditions.

We begin with an analysis of the agent-environment simulation for $\mathbb{SGS}$ systems. We then define the model checking procedure, and we exemplify it on the Train-Gate-Controller. Finally, we prove its soundness. As with the analysis of the $\mathbb{SMR}$ class, we assume, without loss of generality, that for each action $a \in Act_{\mathcal{E}}$ we have that $|\{l \in L_{\mathcal{E}} : a \in P_{\mathcal{E}}(l)\}| = 1$; i.e., an environment's action is enabled at exactly one template state.

6.3.1 Agent-environment simulation

Similarly to the $\mathbb{SMR}$ class, the concrete environment for the $\mathbb{SGS}$ class implements a mutual exclusion controller governing the access to shared resources. This is modelled by means of the environment's looping behaviour as per Lemma 6.1. As we show below, an analogous lemma holds for the $\mathbb{SGS}$ class. However, an environment loop is guaranteed to occur only in path sections that do not contain global-synchronous actions. A subsequence of a path π that does not contain GS actions is said to be a *GS-free section* of π. A GS-free section $\pi(i), \dots, \pi(j)$ is said to be maximal if $\pi(i-1, Act) \in GS$ whenever $i > 1$, and $\pi(j, Act) \in GS$.

Lemma 6.4. *Let $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V}) \in \mathbb{SGS}$ be a PIIS with $\mathcal{T} \leq_{aes} \mathcal{E}$. Let π be a maximal GS-free section in $\mathcal{S}(\overline{n})$. If $ELC(\pi(d))$ for some $d > 1$, then $[d]\pi$ is an environment loop.*

Proof. Assume $ELC(\pi(d))$ for some $d > 1$. Given π is a maximal GS-free section, it follows that: (i) π does not contain GS actions; (ii) $tls_i^j(\pi(1)) \sim_{aes} ls_E(\pi(1))$ for each $(i,j) \in \mathcal{A}(\overline{n})$. Therefore, Lemma 6.1 applies to π, and therefore $[d]\pi$ is an environment loop. □

Intuitively, a global-synchronous action determines a subclass of the shared resources the agents can access. Upon performing a global-synchronous action, the system enters a GS-free section in which the agents can access the shared resources associated with said action in the same way described for $\mathbb{SMR}$ systems.

6.3.2 The SGS procedure

The model checking procedure SGS for $\mathbb{SGS}$ systems is defined by Algorithm 4. Given a PIIS $\mathcal{S} \in \mathbb{SGS}$ and an $\overline{m}$-indexed formula $\forall_{\overline{v}}\phi(\overline{v})$, the procedure first establishes whether or not $\mathcal{T} \leq_{aes} \mathcal{E}$. Upon a successful simulation test, the procedure calculates the cutoff for the given system and specification.

Algorithm 4 Parameterised model checking procedure for $\mathbb{SGS}$ systems.

```
1: procedure SGS($\mathcal{S}$,$\forall_{\overline{v}}\phi(\overline{v})$)
2:     if $\mathcal{T} \leq_{aes} \mathcal{E}$ then
3:         $\bar{c} = cutoff\_SGS(\mathcal{S}, \bar{m})$;
4:         if $\mathcal{S}(\bar{c}) \models \phi[trivial]$ then
5:             return $true$;
6:         else
7:             return $false$;
8:         end if
9:     end if
10: end procedure
```

The cutoff function *cutoff_SGS* maps a PIIS $\mathcal{S}$ and tuple $\overline{m}$ of natural numbers into a tuple $\bar{c}$ of natural numbers that corresponds to the cutoff for $\overline{m}$-indexed formulae. The function is defined in terms of $\overline{m}$.

Definition 6.7 (Cutoff function for $\mathbb{SGS}$ systems)**.** *The cutoff function $cutoff\_SGS$ is defined for $\mathbb{SGS}$ systems as follows:*

$$cutoff\_SGS(\mathcal{S}, \overline{m}) = (\max(1, \overline{m}(1)), \ldots, \max(1, \overline{m}(k))),$$

for any $\mathcal{S} \in \mathbb{SGS}$ with $k \geq 1$ roles and any $\overline{m} \in \mathbb{N}^k$.

Note that differently from $cutoff\_SMR$, $cutoff\_SGS$ depends on the given system only in terms of the number of roles specified for the system. Following the cutoff calculation, the model checking procedure checks the concrete system $\mathcal{S}(\overline{c})$ against the trivial instantiation $\phi[trivial]$ of $\forall_{\overline{v}}(\phi(\overline{v}))$. If $\phi[trivial]$ is satisfied by $\mathcal{S}(\overline{c})$, then the procedure returns true, otherwise it returns false. We assess the soundness of the SGS procedure in Section 6.3.4. First, we exemplify it on the Train-Gate-Controller.

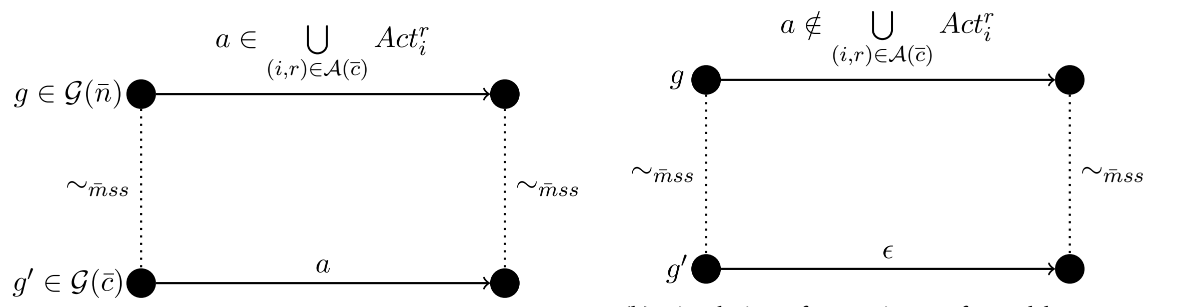
6.3.3 Verifying the Train-Gate-Controller

In section 3.2.1 we represented the Train-Gate-Controller as a PIIS $\mathcal{S}_{TGC}$ composed of an agent template PT encoding prioritised trains, an agent template NT encoding normal trains, and an environment template $\mathcal{E}$ encoding the controller. In section 3.3 we expressed the property "whenever a train is in the tunnel, it knows that no other train is in the tunnel at the same time" in the following $(2,2)$-indexed formula:

$$\begin{aligned}\phi_{TGC} = \forall_{(\{u,v\},\{x,y\})} AG\,(((pt,u) \rightarrow K^{u}_{PT}\,(\neg(pt,v) \wedge \neg(nt,x)))\\ \wedge\,((nt,x) \rightarrow K^{x}_{NT}\,(\neg(nt,y) \wedge \neg(pt,u)))),\end{aligned}$$

where u, v are variables of PT, x, y are variables of NT, the atomic proposition pt holds in the template states in which the template prioritised train is in the tunnel, and the atomic proposition nt holds in the template states in which the template normal train is in the tunnel.

We now use the SMR procedure to establish whether or not the train-gate-controller meets the above specification. To do this, we first observe that $PT \leq_{aes} \mathcal{E}$ and $NT \leq_{aes} \mathcal{E}$. Thus, we can


$a \in \bigcup_{(i,r)\in\mathcal{A}(\overline{c})} Act_i^r$
$g \in \mathcal{G}(\overline{n})$
$\sim_{\overline{m}ss}$
$\sim_{\overline{m}ss}$
$g' \in \mathcal{G}(\overline{c})$
a
$a \notin \bigcup_{(i,r)\in\mathcal{A}(\overline{c})} Act_i^r$
g
$\sim_{\overline{m}ss}$
$\sim_{\overline{m}ss}$
g'
ϵ

(a) Simulation of an action performed by an agent in $\mathcal{A}(\overline{c})$.

(b) Simulation of an action performed by an agent not in $\mathcal{A}(\overline{c})$.

Figure 6.5: The $\overline{m}$-stuttering simulation of a bigger system $\mathcal{S}(\overline{n})$ by the cutoff system $\mathcal{S}(\overline{c})$.

use the cutoff function to calculate a cutoff:

$$\overline{c} = cutoff\_SGS(\mathcal{S}_{TGC}, (2,2)) = (2,2).$$

Therefore, we need to check whether or not $\mathcal{S}_{TGC}\,((2,2)) \models \phi_{TGC}[trivial]$. This check, if mechanised on a standard checker, would return true thereby establishing the correctness of the protocol irrespectively of the number of agents present.

Note that the cutoff identified by `SMR` is smaller than the one identified by `PIIS`. Generally, `PIIS` makes no guarantees on the size of the cutoff, whereas `SMR` is bound to consider the minimal cutoff.

6.3.4 Proof of soundness

Theorem 6.2. *Let $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V}) \in \mathbb{SGS}$ be a PIIS. Let $\forall_{\overline{v}}\phi(\overline{v})$ be an $\overline{m}$-indexed formula. If $\mathcal{T} \leq_{aes} \mathcal{E}$, then* $\texttt{SGS}(\mathcal{S}, \forall_{\overline{v}}\phi(\overline{v}))$ *returns true iff $\mathcal{S} \models \forall_{\overline{v}}\phi(\overline{v})$.*

To prove this result we firstly observe that by the definition of the PMCP and Theorem 3.1, it suffices to show that if $\mathcal{T} \leq_{aes} \mathcal{E}$, then : (i) the cutoff system $\mathcal{S}(\overline{c})$ $\overline{m}$-stuttering simulates every bigger system; (ii) every bigger system $\overline{m}$-stuttering simulates $\mathcal{S}(\overline{c})$. We first show the former.

Part A: The cutoff system $\overline{m}$-stuttering simulates every bigger system

Consider an arbitrarily big system $\mathcal{S}(\overline{n})$ with $\overline{n} \geq \overline{c}$, where $\mathcal{S}(\overline{c})$ is the cutoff system. To simulate an action performed by the agents in $\mathcal{A}(\overline{c})$ in $\mathcal{S}(\overline{n})$, $\mathcal{S}(\overline{c})$ performs the same action. To simulate an action performed by an agent not in $\mathcal{A}(\overline{c})$ in $\mathcal{S}(\overline{n})$, $\mathcal{S}(\overline{c})$ performs the null action. Similarly to the simulation defined in Lemma 6.2 for $\mathbb{SMR}$ systems, the latter does not block potential synchronisations of the environment with the agents in $\mathcal{A}(\overline{c})$ at a later a point. To see this, note that a sequence of actions that are not admitted in the repertoire of the agents in $\mathcal{A}(\overline{c})$ happens only in a GS-free section. Thus, if, for example, $g^1 a^1 g^2 \dots a^{x-1} g^x$ is a GS-free section in $\mathcal{S}(\overline{n})$, where $a^1, a^{x-1} \in \bigcup_{(i,j) \in \mathcal{A}(\overline{c})} AE_i^j$ and $a^2, \dots, a^{x-2} \notin \bigcup_{(i,j) \in \mathcal{A}(\overline{c})} Act_i^j$, then both g^2 and g^{x-1} satisfy the environment loop condition. Therefore, the state of the environment in g^2 is the same as its state in g^{x-1}. Intuitively, whenever an agent not in $\mathcal{A}(\overline{c})$ in $\mathcal{S}(\overline{n})$ takes the lock on a resource via synchronising with the environment, it has to release the resource via the occurrence of an environment loop before an agent in $\mathcal{A}(\overline{c})$ can take the lock on a resource.

Figure 6.5 shows the $\overline{m}$-stuttering simulation between $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{c})$.

Lemma 6.5. *Let $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V}) \in \mathbb{SGS}$ be a PIIS of k roles with $\mathcal{T} \leq_{aes} \mathcal{E}$. Let $\overline{m} \in \mathbb{N}^k$. Then, $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{c})$ for all $\overline{n} \geq \overline{c}$, where $\overline{c} = cutoff\_SGS(\mathcal{S}, \overline{m})$.*

Proof. Let $\overline{n} \geq \overline{c}$ be arbitrary. We show that $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{c})$. Define $\sim_{\overline{m}ss} = R_1 \cap R_2 \subseteq \mathcal{G}(\overline{n}) \times \mathcal{G}(\overline{c})$ as follows:

$$(g, g') \in R_1 \text{ iff } ls_i^r(g) = ls_i^r(g'), \text{ for } (i, r) \in \mathcal{A}(\overline{c})$$

$$(g, g') \in R_2 \text{ iff } g \rightarrow_{X*} g^1 \rightarrow_a g^2 \text{ implies that } ls_E(g^1) = ls_E(g'), \text{ where}$$

$$a \in \bigcup_{(i,r) \in \mathcal{A}(\overline{c})} AE_i^r \cup GS \text{ and } X = \bigcup_{(i,r) \in \mathcal{A}(\overline{c})} A_i^r \cup \bigcup_{(i,r) \notin \mathcal{A}(\overline{c})} Act_i^r$$

We show that $\sim_{\overline{m}ss}$ is an $\overline{m}$-stuttering simulation relation between $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{c})$.

Clearly, $(\iota(\overline{n}), \iota(\overline{c})) \in R_1$. To show that $(\iota(\overline{n}), \iota(\overline{c})) \in R_2$, suppose that $\iota(\overline{n}) \rightarrow_{X*} g^1 \rightarrow_a g^2$, where $a \in \bigcup_{(i,r) \in \mathcal{A}(\overline{c})} AE_i^r \cup GS$ and $X = \bigcup_{(i,r) \in \mathcal{A}(\overline{c})} A_i^r \cup \bigcup_{(i,r) \notin \mathcal{A}(\overline{c})} Act_i^r$. We have to show that $ls_E(g^1) = ls_E(\iota(\overline{c}))$. If the path $\iota(\overline{n}), \dots, g^1$ consists only of asynchronous actions, then the case follows trivially. Otherwise, there must be an agent-environment action in $\iota(\overline{n}), \dots, g^1$.

In this case, we have $ELC(g^1)$. By Lemma 6.4, $\iota(\overline{n}), \ldots, g^1$ is an environment loop. So, as $ls_E(\iota(\overline{n})) = ls_E(\iota(\overline{c}))$, $ls_E(g^1) = ls_E(\iota(\overline{c}))$. Therefore, $(\iota(\overline{n}), \iota(\overline{c})) \in R_2$, and therefore, $\iota(\overline{n}) \sim_{\overline{m}ss} \iota(\overline{c})$.

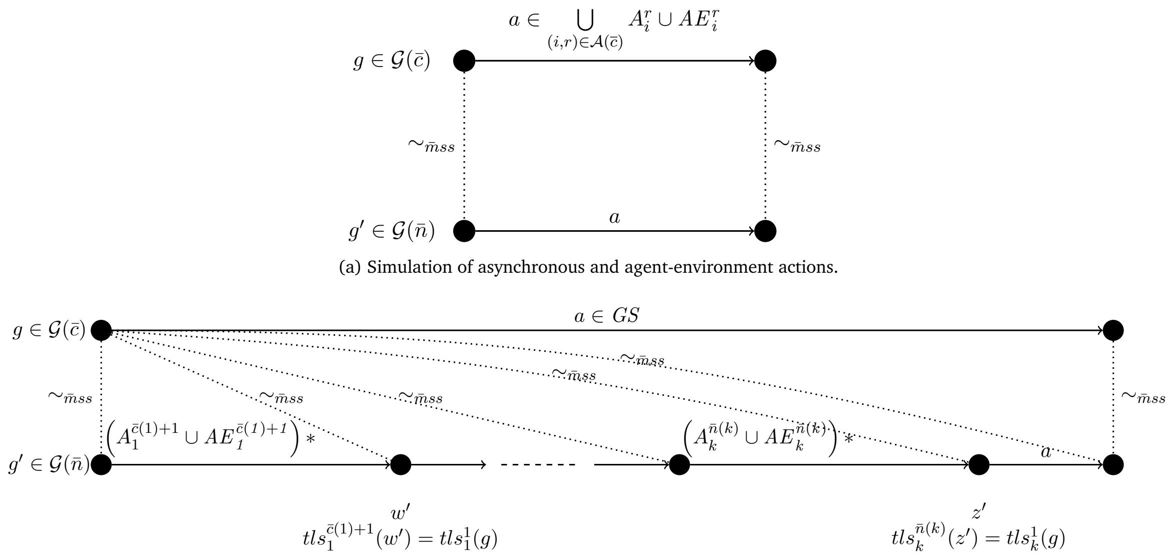
Let $g \sim_{\overline{m}ss} g'$ for an arbitrary pair of global states in $\mathcal{G}(\overline{n}) \times \mathcal{G}(\overline{c})$. We show the simulation requirements (i) and (iii). Requirement (i) follows by the definition of R_1. To show requirement (iii), let $\pi \in \Pi(g)$. Define a path $\pi' \in \Pi(g')$ as $\pi' = \pi'(1)\pi'(1, Act)\pi'(2, Act)\ldots$, where $\pi'(x, Act)$ is defined for each $x \geq 1$ as follows:

- $\pi'(x, Act) = \pi(x, Act)$ if $\pi(x, Act) \in \bigcup_{(i,r)\in\mathcal{A}(\overline{c})} Act_i^r$;
- otherwise, $\pi'(x, Act) = \epsilon$.

Partition π and π' into singleton blocks. We use induction on the length of π to show that: (i) π' is a valid path; (ii) $\pi(y) \sim_{\overline{m}ss} \pi'(y)$ for each $y \geq 1$. The base step for π of length 1 follows trivially. For the inductive step, assume that $[x-1]\pi' \in \Pi(g')$ and $\pi(y) \sim_{\overline{m}ss} \pi'(y)$ for each y with $1 \leq y \leq x-1$. We show that $[x]\pi' \in \Pi(g')$ and $\pi(x) \sim_{\overline{m}ss} \pi'(x)$. The latter follows by the same argument used to show earlier that $\iota(\overline{n}) \sim_{\overline{m}ss} \iota(\overline{c})$. To establish the former, we consider cases on the type of the action $\pi'(x-1, Act)$. The case where $\pi'(x-1, Act)$ is the ϵ action, or an asynchronous action follows trivially. Assume that $\pi'(x-1, Act) \in \bigcup_{(i,r)\in\mathcal{A}(\overline{c})} AE_i^r \cup GS$. We have to show that the action is enabled by the protocol of the agents and the protocol of the environment. The former is clear by the definition of R_1. In the following we show the latter.

Define an integer d as follows. If there is no global-synchronous action in $[x-1]\pi$, then $d = 1$. Otherwise, $d = d' + 1$, where d' is the largest integer with $1 \leq d' \leq x-1$ that satisfies $\pi(d', Act) \in GS$. By definition of d, $\pi(d), \ldots, \pi(x-1)$ is a GS-free section. We have two cases.

1. $\pi(d), \ldots, \pi(x-1)$ contains an action in $\bigcup_{(i,r)\in\mathcal{A}(\overline{c})} AE_i^r$. In this case, let z be the greatest integer with $d \leq z < x-1$ such that $\pi(z, Act) \in \bigcup_{(i,r)\in\mathcal{A}(\overline{c})} AE_i^r$. We show that $ls_E(\pi(z+1)) = ls_E(\pi(x-1))$. The claim follows trivially when $\pi(z+1), \ldots, \pi(x-1)$ does not contain an agent-environment action. For the case where $\pi(z+1), \ldots, \pi(x-1)$ contains an agent-environment action, we have that $ELC(\pi(x-1))$. Thus, by Lemma 6.4, we



$a \in \bigcup_{(i,r)\in\mathcal{A}(\bar{c})} A_i^r \cup AE_i^r$
$g \in \mathcal{G}(\bar{c})$
$\sim_{\bar{m}ss}$
$\sim_{\bar{m}ss}$
a
$g' \in \mathcal{G}(\bar{n})$

(a) Simulation of asynchronous and agent-environment actions.

$a \in GS$
$g \in \mathcal{G}(\bar{c})$
$\sim_{\bar{m}ss}$
$g' \in \mathcal{G}(\bar{n})$
$\left(A_1^{\bar{c}(1)+1} \cup AE_1^{\bar{c}(1)+1}\right)*$
$\left(A_k^{\bar{n}(k)} \cup AE_k^{\bar{n}(k)}\right)*$
a
w'
$tls_1^{\bar{c}(1)+1}(w') = tls_1^1(g)$
z'
$tls_k^{\bar{n}(k)}(z') = tls_k^1(g)$

(b) Simulation of global-synchronous actions.

Figure 6.6: the $\overline{m}$-stuttering simulation of the cutoff system $\mathcal{S}(\overline{c})$ by a bigger system $\mathcal{S}(\overline{n})$.

obtain $ls_E(\pi(z+1)) = ls_E(\pi(x-1))$. By consrtuction of π', $ls_E(\pi(z+1)) = ls_E(\pi'(z+1))$ and $ls_E(\pi'(z+1)) = ls_E(\pi'(x-1))$. Therefore, $ls_E(\pi'(x-1)) = ls_E(\pi(x-1))$. Hence, $\pi'(x-1, Act) \in P_E(\pi'(x-1))$.

2. $\pi(d), \ldots, \pi(x-1)$ does not contain an action in $\bigcup_{(i,r)\in\mathcal{A}(\overline{c})} AE_i^r$. By the inductive hypothesis, $(\pi(d), \pi'(d)) \in R_2$. Therefore, $\pi(d) \rightarrow_{X*} \pi(x-1) \rightarrow_a \pi(x)$ for some $a \in \bigcup_{(i,r)\in\mathcal{A}(\overline{c})} AE_i^r \cup GS$. Consequently, $ls_E(\pi(x-1)) = ls_E(\pi'(d))$. Therefore, by construction of π', $ls_E(\pi(x-1)) = ls_E(\pi'(x-1))$. It follows that $\pi(x-1, Act) \in P_E(\pi'(x-1))$.

Consequently, π' is a valid path, and $\pi(y) \sim_{\overline{m}ss} \pi'(y)$ for each $y \geq 1$. Therefore, $\sim_{\overline{m}ss}$ satisfies simulation requirement (iii). As (g, g') was arbitrary, $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{c})$ as required. □

We have thus shown that the cutoff system $\overline{m}$-stuttering simulates every bigger system. We conclude the proof for Theorem 6.2 by showing that every bigger system $\overline{m}$-stuttering simulates the cutoff system.

Part B: Every bigger system $\overline{m}$-stuttering simulates the cutoff system

Figure 6.6 shows the $\overline{m}$-stuttering simulation between the cutoff system $\mathcal{S}(\overline{c})$ and a bigger system $\mathcal{S}(\overline{n})$. To simulate an asynchronous or an agent-environment action of $\mathcal{S}(\overline{c})$, $\mathcal{S}(\overline{n})$

performs the same action. Consider a global-synchronous action performed in $\mathcal{S}(\overline{c})$. Clearly, every agent in $\mathcal{S}(\overline{c})$ is at a state enabling the global-synchronous action. $\mathcal{S}(\overline{n})$ simulates a global-synchronous action as follows. One by one, every agent $(i,j) \in \mathcal{A}(\overline{n}) \setminus \mathcal{A}(\overline{c})$ in $\mathcal{S}(\overline{n})$ mimicks agent $(i,1)$ by performing the same sequence of actions, thereby reaching the local state of agent $(i,1)$ in which the global-synchronous action is enabled. Then, $\mathcal{S}(\overline{n})$ performs the global-synchronous action. The following lemma formally defines this simulation and shows it to be an $\overline{m}$-stuttering simulation.

Lemma 6.6. *Let $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V}) \in \mathbb{SGS}$ be a PIIS of k roles with $\mathcal{T} \leq_{aes} \mathcal{E}$. Let $\overline{m} \in \mathbb{N}^k$. Then, $\mathcal{S}(\overline{c}) \leq_{\overline{m}ss} \mathcal{S}(\overline{n})$ for all $\overline{n} \geq \overline{c}$, where $\overline{c} = cutoff\_SGS(\mathcal{S}, \overline{m})$.*

Proof. Let $\overline{n} \geq \overline{c}$. Assume an integer i with $1 \leq i \leq k$. Define $\overline{n}'$ to be a k-tuple of integers such that: $\overline{n}'(i) = \overline{n}(i) + 1$; $\overline{n}'(j) = \overline{n}(j)$ for all j with $1 \leq j \leq k$ and $j \neq i$. We show that $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{n}')$. That is, we prove that the system obtained by adding one concrete agent (from an arbitrary template) to $\mathcal{S}(\overline{n})$ $\overline{m}$-stuttering simulates $\mathcal{S}(\overline{n})$. The inductive application of the latter shows the main claim of the lemma.

Let $r = \overline{n}'(i)$. The simulation relation $\sim_{\overline{m}ss} = R_1 \cap R_2 \cap R_3 \subseteq \mathcal{G}(\overline{n}) \times \mathcal{G}(\overline{n}')$ is defined by the following.

$$(g, g') \in R_1 \text{ iff } ls_j^q(g) = ls_j^q(g') \text{ for } (j,q) \in \mathcal{A}(\overline{n})$$

The above ensures that the local state of each agent in $\mathcal{S}(\overline{n})$ is equal to its local state in a related global state in $\mathcal{S}(\overline{n}')$. The relations R_2 and R_3 specify the local state of the environment and the local state of agent (i,r) in related global states. Specifically, R_2 and R_3 are defined to induce a mimicking behaviour on agent (i,r) in $\mathcal{S}(\overline{n}')$ w.r.t agent $(i,1)$ in $\mathcal{S}(\overline{n})$.

$$(g, g') \in R_2 \text{ iff } g \rightarrow_{X*} g^1 \rightarrow_{GS} g^2 \text{ implies that } g' \rightarrow_{Y*} g'^1, \text{where}$$
$$X = \bigcup_{(j,q) \in \mathcal{A}(\overline{n})} A_j^q \cup AE_j^q,\ Y = A_i^r \cup AE_i^r,\ tls_i^r(g'^1) = tls_i^1(g^1),\ \text{and } ls_E(g'^1) = ls_E(g^1)$$

If $(g, g') \in R_2$ and a state g^1 is reachable from g at which a GS action is enabled, then agent (i,r) in $\mathcal{S}(\overline{n}')$ is able to change its local state to the local state of agent $(i,1)$ in $\mathcal{S}(\overline{n})$ via asynchronous and agent-environment transitions. Thus, whenever a GS-transition is taken in $\mathcal{S}(\overline{n})$, agent (i,r) is able to move to a local state at which the GS action is enabled. Note that $(R_1 \cap R_2)$-related states may disagree on the environment's local state. Because of this, given an arbitrary pair of $(R_1 \cap R_2)$-related states, $\mathcal{S}(\overline{n}')$ may not be able to simulate $\mathcal{S}(\overline{n})$. To circumvent this, we define R_3 to ensure that whenever $(R_1 \cap R_2)$-related states disagree on the environment's state, agent (i, r) can cause the environment to appropriately change its state.

$$(g, g') \in R_3 \text{ iff } ls_E(g) \neq ls_E(g') \text{ implies that } g' \rightarrow_{Y*} g'^1, \text{where}$$
$$Y = A_i^r \cup AE_i^r, tls_i^r(g'^1) = tls_i^1(g), \text{ and } ls_E(g'^1) = ls_E(g)$$

We show that $\sim_{\overline{m}ss}$ is an $\overline{m}$-stuttering simulation relation between $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{n}')$. To do this, we first show that $(\iota(\overline{n}), \iota(\overline{n}')) \in \sim_{\overline{m}ss}$. It is obvious that $(\iota(\overline{n}), \iota(\overline{n}')) \in R_1 \cap R_3$. To show that $(\iota(\overline{n}), \iota(\overline{n}')) \in R_2$, suppose that $\iota(\overline{n}) \rightarrow_{X*} g^1 \rightarrow_{GS} g^2$. We have to show that $\iota(\overline{n}') \rightarrow_{Y*} g'^1$, where $tls_i^r(g'^1) = tls_i^1(g^1)$ and $ls_E(g'^1) = ls_E(g^1)$. Let θ be the sequence of actions of agent $(i, 1)$ in the path $\pi = \iota(\overline{n}), \ldots, g^1$. Define θ' to be the sequence θ, but with each action indexed with r instead of 1. Let $\pi' = \iota(\overline{n}') \circ \theta'$ be a path in $\mathcal{S}(\overline{n}')$. It should be clear that π' is a valid path if it consists only of asynchronous actions. If this is not the case, then observe that for every two successive synchronisations, say in states $\pi(d)$ and $\pi(d')$, of agent $(i, 1)$ with the environment in π, if there is a different agent that synchronises with the environment in the path from $\pi(d)$ to $\pi(d')$, then the local state of the environment is the same in $\pi(d)$ and $\pi(d')$. This follows by Lemma 6.4 which gives $ELC(\pi(d))$ and $ELC(\pi(d'))$. So, the environment allows for the actions in θ' to be performed. It follows that $\iota(\overline{n}') \rightarrow_{Y*} \pi'[]$, where $tls_i^r(\pi'[]) = tls_i^1(g^1)$ and $ls_E(\pi'[]) = ls_E(g^1)$.

Now assume an arbitrary pair $(g, g') \in \mathcal{G}(\overline{n}) \times \mathcal{G}(\overline{n}')$ of global states such that $g \sim_{\overline{m}ss} g'$. We show the simulation requirements (i) and (iii). Requirement (i) follows by the definition of R_1. Consider requirement (iii) and a path $\pi \in \Pi(g)$. We construct a path $\pi' \in \Pi(g')$ as required by the $\overline{m}$-stuttering-simulation. The construction is inductive on the length of π. Assume that, for a prefix $[x-1]\pi$, we have already constructed a prefix $[x'-1]\pi'$, a partition $B_1, \ldots, B_{y-1}$ of the states in $[x-1]\pi$, and a partition $B'_1, \ldots, B'_{y-1}$ of the states in $[x'-1]\pi'$ into corresponding blocks. Let $\pi(d), \ldots, \pi(x-1)$ be the maximal GS-free section that is a suffix of $[x-1]\pi$. We now define the next blocks B_y and B'_y. We have two cases depending on the equality of the environment's local state in $\pi(x-1)$ and $\pi'(x'-1)$.

1. $ls_E(\pi(x-1)) = ls_E(\pi'(x'-1))$. If $\pi(x-1, Act) \in X$, then define $B_y = \pi(x)$ and $B'_y = \pi'(x')$, where $\pi'(x'-1) \rightarrow_{\pi(x-1,Act)} \pi'(x')$. It is obvious that the latter is a valid transition and that $\pi(x) \sim_{\overline{m}ss} \pi'(x')$.

   Otherwise, assume that $\pi(x-1, Act) \in GS$. By $(\pi(x-1), \pi'(x'-1)) \in R_2$, we have that $\pi'(x'-1) \rightarrow_{Y*} \pi'(x'+d)$, where $d \geq 0$, $tls_i^r(\pi'(x'+d)) = tls_i^1(\pi(x-1))$ and $ls_E(\pi'(x'+d)) = ls_E(\pi(x-1))$. Therefore, $\pi(x-1, Act)$ is enabled at $\pi'(x'+d)$. Extend B'_{y-1} to $B'_{y-1} \circ \pi'(x'), \ldots, \pi'(x'+d)$. Define $B_y = \pi(x)$ and $B'_y = \pi'(x'+d+1)$, where $\pi'(x'+d) \rightarrow_{\pi(x-1,Act)} \pi'(x'+d+1)$. We get that $\pi(x-1)$ is $\sim_{\overline{m}ss}$-related to every state in B'_{y-1}. Additionally, $(\pi(x), \pi'(x'+d+1)) \in R_1 \cap R_3$. Moreover, the argument used earlier to show that $(\iota(\overline{n}), \iota(\overline{n}')) \in R_2$ can be used here to show that $(\pi(x), \pi'(x'+d+1)) \in R_2$. Hence, $\pi(x) \sim_{\overline{m}ss} \pi'(x'+d+1)$.

2. $ls_E(\pi(x-1)) \neq ls_E(\pi'(x'-1))$. In this case, from $(\pi(x-1), \pi'(x'-1)) \in R_3$, it follows that $\pi'(x'-1) \rightarrow_{Y*} \pi'(x'+d)$, where $d \geq 0$, $tls_i^r(\pi'(x'+d)) = tls_i^1(\pi(x-1))$ and $ls_E(\pi'(x'+d)) = ls_E(\pi(x-1))$. Therefore, $\pi(x-1, Act)$ is enabled in $\pi'(x'+d)$. Extend B'_{y-1} to $B'_{y-1} \circ \pi(x'), \ldots, \pi'(x'+d)$. Define $B_y = \pi(x)$ and $B'_y = \pi'(x'+d+1)$, where $\pi'(x'+d) \rightarrow_{\pi(x-1,Act)} \pi'(x'+d+1)$. We get that $\pi(x-1)$ is $\sim_{\overline{m}ss}$-related to every state in B'_{y-1}. Additionally, $(\pi(x), \pi'(x'+d+1)) \in R_1 \cap R_3$. Moreover, the argument used earlier to show that $(\iota(\overline{n}), \iota(\overline{n}')) \in R_2$ can be used here to show that $(\pi(x), \pi'(x'+d+1)) \in R_2$. Hence, $\pi(x) \sim_{\overline{m}ss} \pi'(x'+d+1)$.

Having shown simulation requirements (i) and (iii) for an arbitrary pair of global states, it follows that $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{n}')$. By the inductive application of the latter and by transitivity of $\leq_{\overline{m}ss}$, $\mathcal{S}(\overline{c}) \leq_{\overline{m}ss} \mathcal{S}(\overline{n})$. □

Corollary 6.2. *Let $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V}) \in \mathbb{SGS}$ be a PIIS of k roles with $\mathcal{T} \leq_{aes} \mathcal{E}$. Let $\forall_{\overline{v}}\phi(\overline{v})$ be an $\overline{m}$-indexed formula . Then, $\mathcal{S}(\overline{c}) \models \forall_{\overline{v}}\phi(\overline{v})$ iff $\forall \overline{n} \geq \overline{c}. \mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v})$, where $\overline{c} = cutofff\_SGS(\mathcal{S}, \overline{m})$.*

Proof. Let $\overline{n} \geq \overline{c}$ be arbitrary. By Theorem 3.1 it suffices to show that $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{c})$ and $\mathcal{S}(\overline{c}) \leq_{\overline{m}ss} \mathcal{S}(\overline{n})$. The former is given by Lemma 6.5. The latter is given by Lemma 6.6. □

This concludes the proof of Theorem 6.2. In contrast with the $\mathbb{SMR}$ class where cutoffs depend on the cardinality of the action dependency sets, Corollary 6.2 provides a methodology for solving the PMCP by giving the conditions under which the problem can be solved by checking only the cutoff system. This clearly has considerable advantages in applications as section 6.3.3 demonstrates.

6.4 Verifying $\mathbb{SFE}$ Systems

We finally study the $\mathbb{SFE}$ class of systems defined on asynchronous, role-synchronous, and global-synchronous actions. In contrast with $\mathbb{SMR}$ and $\mathbb{SGS}$, the existence of cutoffs in the $\mathbb{SFE}$ class does not depend on the existence of an agent-environment simulation between the agent and environment templates. In particular, we show that a cutoff exists for any given system and any given specification. This enables us to define a sound and complete model checking procedure. We begin with the definition of the model checking procedure which we exemplify it on the autonomous robot scenario. We then show it to be sound.

6.4.1 The `SFE` procedure

Algorithm 5 Parameterised model checking procedure for $\mathbb{SFE}$ systems.

```
1: procedure SFE($\mathcal{S}$,$\forall_{\bar{v}}\phi(\bar{v})$)
2:     $\bar{c} = cutoff\_SFE(\mathcal{S}, \bar{m})$;
3:     if $\mathcal{S}(\bar{c}) \models \phi[trivial]$ then
4:         return $true$;
5:     else
6:         return $false$;
7:     end if
8: end procedure
```

The model checking procedure for $\mathbb{SFE}$ systems is defined by Algorithm 5. Given a PIIS $\mathcal{S} \in \mathbb{SFE}$ and an $\overline{m}$-indexed formula $\forall_{\overline{v}}\phi(\overline{v})$, the procedure calculates the cutoff for the given system and specification. The cutoff function *cutoff_SFE* maps a PIIS $\mathcal{S}$ and a tuple $\overline{m}$ of natural numbers into a tuple $\bar{c}$ of natural numbers that corresponds to the cutoff for $\overline{m}$-indexed formulae. Identically to *cutoff_SGS* and differently from *cutoff_SMR*, *cutoff*_$\mathbb{SFE}$ depends on the system under analysis only in terms of $\overline{m}$.

Definition 6.8 (Cutoff function for $\mathbb{SFE}$ systems)**.** *The cutoff function $cutoff\_SFE$ is defined for $\mathbb{SFE}$ systems as follows.*

$$cutoff\_SFE(\mathcal{S}, \overline{m}) = (\max(1, \overline{m}(1)), \ldots, \max(1, \overline{m}(k))) \,,$$

for any $\mathcal{S} \in \mathbb{SFE}$ with $k \geq 1$ roles and any $\overline{m} \in \mathbb{N}^k$.

Following the cutoff calculation, the model checking procedure checks the concrete system $\mathcal{S}(\overline{c})$ against the trivial instantiation $\phi[trivial]$ of $\forall_{\overline{v}}(\phi(v))$. If $\phi[trivial]$ is satisfied by $\mathcal{S}(\overline{c})$, then the procedure returns *true*, otherwise it returns *false*. Note that, differently from the model checking procedures for $\mathbb{SMR}$ and $\mathbb{SGS}$ systems, the model checking procedure for $\mathbb{SFE}$ systems always produces an output. We show that this output is always correct in section 6.4.3. First, we compute its value for the autonomous robot example.

6.4.2 Verifying the autonomous robot example

In section 3.2.1 we encoded the autonomous robot example as a PIIS $\mathcal{S}_{AR}$ composed of an agent template $TR1$ representing robots with access to a sensor, an agent template $TR2$ representing robots with no access to a sensor, and an environment template $\mathcal{E}$ representing the environment. In section 3.3 we expressed the property “whenever a robot halts, it knows that it is in the goal region" in the following $(1, 1)$-indexed formula:

$$\phi_{AR} = \forall_{(\{v\},\{u\})} AG((h\_1, v) \rightarrow K^{v}_{TR1}((gr\_1, v)) \wedge (h\_2, u) \rightarrow K^{u}_{TR2}((gr\_2, u)))$$

where v is a variable of $TR1$, u is a variable of $TR2$, the atomic proposition gr_1 (gr_2, respectively) holds in the template states where the value of the position component of template robot 1 (template robot 2, respectively) is in $\{2, 3, 4\}$, and the atomic proposition h_1 (h_2, respectively) holds in the template states where the template robot 1 (template robot 2, respectively) has halted.

We now use the `SFE` procedure to establish whether or not the autonomous robot meets the above specification. We have that

$$\overline{c} = \mathit{cutoff\_SFE}(\mathcal{S}_{AR}, (1,1)) = (1,1).$$

Thus, we need to check whether or not $\mathcal{S}_{AR}((1,1)) \models \phi_{AR}[\mathit{trivial}]$. The latter query can be tested with a standard model checker; this will return true thereby establishing the correctness of the protocol irrespectively of the number of robots present.

6.4.3 Proof of soundness

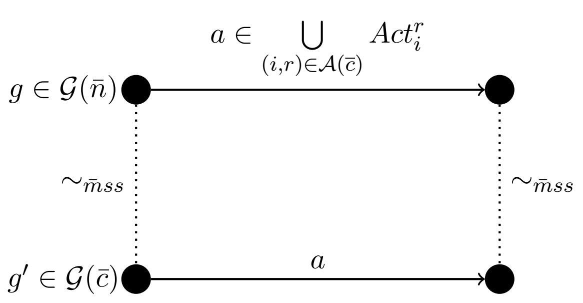
Theorem 6.3. *Let $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V}) \in \mathbb{SFE}$ be a PIIS. Let $\forall_{\overline{v}}\phi(\overline{v})$ be an $\overline{m}$-indexed formula. Then, $\mathit{SFE}(\mathcal{S}, \forall_{\overline{v}}\phi(\overline{v}))$ returns true iff $\mathcal{S} \models \forall_{\overline{v}}\phi(\overline{v})$.*

We prove this result by showing that: (i) the cutoff system $\mathcal{S}(\overline{c})$ $\overline{m}$-stuttering simulates every bigger system; (ii) every bigger system $\overline{m}$-stuttering simulates $\mathcal{S}(\overline{c})$. Differently from the corresponding results for the $\mathbb{SMR}$ and $\mathbb{SGS}$ classes, the absence of agent-environment and multi-role actions in the $\mathbb{SFE}$ class removes the necessity to simulate synchronisations of the environment on actions that are not admitted by the agents in $\mathcal{A}(\overline{m})$. Indeed, whenever an agent not in $\mathcal{A}(\overline{m})$ synchronises with the environment, all agents of the same role (or all agents in the system depending on the type of the action) synchronise the environment.

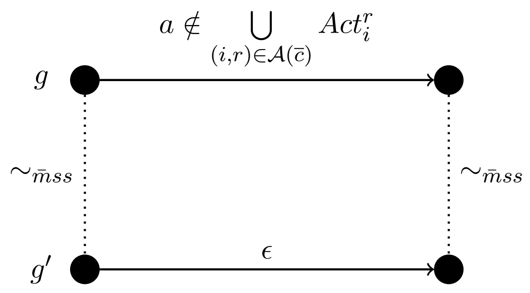
Part A: The cutoff system $\overline{m}$-stuttering simulates every bigger system

Consider an arbitrarily big system $\mathcal{S}(\overline{n})$ with $\overline{n} \geq \overline{c}$, where $\mathcal{S}(\overline{c})$ is the cutoff system. Figure 6.7 shows the $\overline{m}$-stuttering simulation between $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{c})$. To simulate an action performed by the agents in $\mathcal{A}(\overline{c})$ in $\mathcal{S}(\overline{n})$, $\mathcal{S}(\overline{c})$ performs the same action. Any action performed in $\mathcal{S}(\overline{n})$ that is not admitted by an agent in $\mathcal{A}(\overline{c})$ is bound to be an asynchronous action. As only the state of the agent performing the action is updated, $\mathcal{S}(\overline{c})$ simulates these actions by performing the null action.

Lemma 6.7. *Let $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V}) \in \mathbb{SFE}$ be a PIIS of k roles. Let $\overline{m} \geq \mathbb{N}^k$. Then, $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{c})$ for all $\overline{n} \geq \overline{c}$, where $\overline{c} = \mathit{cutoff\_SFE}(\mathcal{S}, \overline{m})$.*


$a \in \bigcup_{(i,r)\in\mathcal{A}(\bar{c})} Act_i^r$
$g \in \mathcal{G}(\bar{n})$
$\sim_{\bar{m}ss}$
$\sim_{\bar{m}ss}$
$g' \in \mathcal{G}(\bar{c})$
a

(a) Simulation of an action performed by an agent in $\mathcal{A}(\bar{c})$.



(b) Simulation of an action performed by an agent not in $\mathcal{A}(\bar{c})$.

Figure 6.7: The $\overline{m}$-stuttering simulation of a bigger system $\mathcal{S}(\overline{n})$ by the cutoff system $\mathcal{S}(\overline{c})$.

Proof. Choose an arbitrary $\overline{n} \geq \overline{c}$. We show that $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{c})$. Define the simulation relation $\sim_{\overline{m}ss} \subseteq \mathcal{G}(\overline{n}) \times \mathcal{G}(\overline{c})$ as follows:

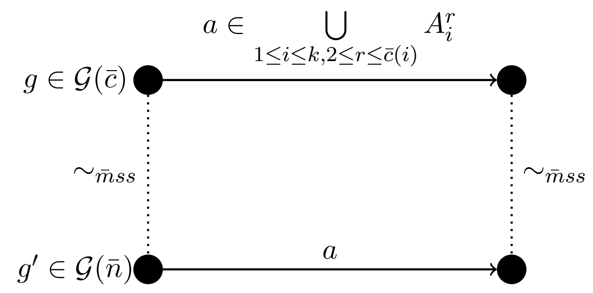
$$(g, g') \in \sim_{\overline{m}ss} \text{ iff } ls_i^r(g) = ls_i^r(g'), \text{ for } (i,r) \in \mathcal{A}(\overline{c})$$

We show that $\sim_{\overline{m}ss}$ is an $\overline{m}$-stuttering simulation between $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{c})$. It is clear that $\iota(\overline{n}) \sim_{\overline{m}ss} \iota(\overline{c})$. Let $g \sim_{\overline{m}ss} g'$ for an arbitrary pair of global states in $\mathcal{G}(\overline{n}) \times \mathcal{G}(\overline{c})$. We show the simulation requirements (i) and (iii). Requirement (i) follows by the definition of $\sim_{\overline{m}ss}$. For simulation requirement (iii), let $\pi \in \Pi(g)$. Construct a path $\pi' \in \Pi(g')$ as follows: for each $j \geq 1$, $\pi'(j, Act) = \pi(j, Act)$ if $\pi(j, Act) \in \bigcup_{(i,r)\in\mathcal{A}(\overline{c})} Act_i^r$; otherwise, $\pi'(j, Act) = \epsilon$. It can be checked that π' is a valid path, and that $\pi(j) \sim_{\overline{m}ss} \pi'(j)$, for each $j \geq 1$. Thus, partition π and π' into singleton blocks. It follows that $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{c})$. $\square$

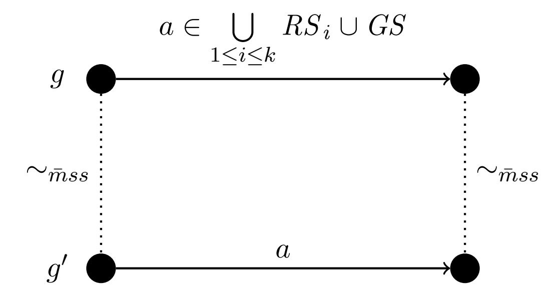
Part B: Every bigger system m̅-stuttering simulates the cutoff system

Figure 6.8 shows the $\overline{m}$-stuttering simulation between $\mathcal{S}(\overline{c})$ and $\mathcal{S}(\overline{n})$. An asynchronous action a of template $\mathcal{T}_i$ is simulated by $\mathcal{S}(\overline{c})$ by means of two cases. If a is not admitted in the repertoire of agent $(i,1)$, then $\mathcal{S}(\overline{n})$ simply performs a. If a is admitted in the repertoire of agent $(i,1)$, then, one by one, the agents $(i,1), (i, \overline{c}(i)+1), \ldots, (i, \overline{n}(i))$ perform a. Thus, for each role i, every agent in $\{\overline{c}(i)+1, \ldots, \overline{n}(i)\}$ mimics agent $(i,1)$. $\mathcal{S}(\overline{n})$ may then simulate role-synchronous actions and global-synchronous actions by performing the action in question.

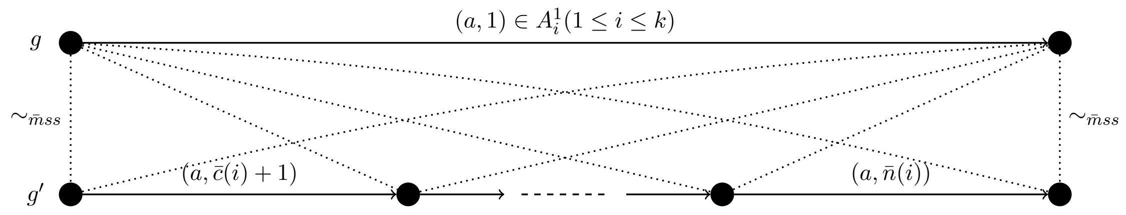
Lemma 6.8. *Let $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V}) \in \mathbb{SFE}$ be a PIIS of k role. Let $\overline{m} \in \mathbb{N}^k$. Then, $\mathcal{S}(\overline{c}) \leq_{\overline{m}ss} \mathcal{S}(\overline{n})$, where $\overline{c} = cutoff\_SFE(\mathcal{S}, \overline{m})$.*


$a \in \bigcup_{1 \le i \le k, 2 \le r \le \bar{c}(i)} A_i^r$
$g \in \mathcal{G}(\bar{c})$
$\sim_{\bar{m}ss}$
$\sim_{\bar{m}ss}$
$g' \in \mathcal{G}(\bar{n})$
a

(a) Simulation of asynchronous actions of agents with index different than 1.



(b) Simulation of role-synchronous and global-synchronous actions.



(c) Simulation of asynchronous actions of agents with index equal to 1.

Figure 6.8: The $\overline{m}$-stuttering simulation of the cutoff system $\mathcal{S}(\overline{c})$ by a bigger system $\mathcal{S}(\overline{n})$.

Proof. Let $\overline{n} \geq \overline{c}$. Assume an integer i with $1 \leq i \leq k$. Define $\overline{n}'$ to be a k-tuple of integers such that: $\overline{n}'(i) = \overline{n}(i) + 1$; $\overline{n}'(j) = \overline{n}(j)$ for all j with $1 \leq j \leq k$ and $j \neq i$. We show that $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{n}')$. That is, we prove that the system obtained by adding one concrete agent (from an arbitrary template) to $\mathcal{S}(\overline{n})$ $\overline{m}$-stuttering simulates $\mathcal{S}(\overline{n})$. The inductive application of the latter shows the main claim of the lemma.

Assume $r = \overline{n}'(i)$. Define the simulation relation $\sim_{\overline{m}ss} \subseteq \mathcal{G}(\overline{n}) \times \mathcal{G}(\overline{n}')$ as follows:

$$g \sim_{\overline{m}ss} g' \text{ iff } ls_j^q(g) = ls_j^q(g') \text{ for } (j,q) \in \mathcal{A}(\overline{n}), ls_E(g) = ls_E(g'), \text{ and}$$
$$g' \rightarrow_{A_i^r *} g'^1, \text{ where } tls_i^r(g'^1) = tls_i^1(g')$$

If $g \sim_{\overline{m}ss} g'$, then the local states of the agents in $\mathcal{A}(\overline{n})$ and the local state of the environment are the same in g and g'. Additionally, the agent (i, r) in $\mathcal{S}(\overline{n}')$ is able to change its local state to the local state of agent $(i, 1)$ in $\mathcal{S}(\overline{n}')$ via asynchronous transitions. We show that $\sim_{\overline{m}ss}$ is an $\overline{m}$-stuttering simulation between $\mathcal{S}(\overline{n})$ and $\mathcal{S}(\overline{n}')$.

The case of $\iota(\overline{n}) \sim_{\overline{m}ss} \iota(\overline{n}')$ follows trivially. Suppose that $g \sim_{\overline{m}ss} g'$ for an arbitrary pair of global states in $\mathcal{G}(\overline{n}) \times \mathcal{G}(\overline{n}')$. We show the simulation requirements (i) and (iii). The former requirement follows by the definition of $\sim_{\overline{m}ss}$. For the latter requirement, let $\pi \in \Pi(g)$. By induction on the length of π, we construct a path $\pi' \in \Pi(g')$ such that: (i) π' is as required by the $\overline{m}$-stuttering simulation; (ii) $tls_i^r(\pi'[]) = tls_i^1(\pi'[])$.

For the base step, $g \sim_{\overline{m}ss} g'$ gives $g' \to_{A_i^r *} g'^1$, where $tls_i^r(g'^1) = tls_i^1(g')$. Let π' be the path from g' to g'^1. The first blocks B_1, B'_1 are defined as follows: $B_1 = g$, and B'_1 is the sequence of states in π'. Clearly, g is $\sim_{\overline{m}ss}$-related to every state in π'.

For the inductive step, assume that for a prefix $[x-1]\pi$ we have already constructed a prefix $[x'-1]\pi'$ such that $tls_i^r(\pi'(x'-1)) = tls_i^1(\pi'(x'-1))$, and a partition of the states in $[x-1]\pi$ and $[x'-1]\pi'$ into corresponding blocks. We now define the next blocks B_x and B'_x. There are two cases.

1. $\pi(x-1, Act) \notin A_i^1$. Define $B_x = \pi(x)$ and $B'_x = \pi'(x')$, where $\pi'(x'-1) \to_{\pi(x-1,Act)} \pi'(x')$. The inductive hypothesis gives $\pi(x-1) \sim_{\overline{m}ss} \pi'(x'-1)$ and $tls_i^r(\pi'(x'-1)) = tls_i^1(\pi'(x'-1))$. Therefore, $\pi'(x'-1) \to_{\pi(x-1,Act)} \pi'(x')$ is a valid transition, $\pi(x) \sim_{\overline{m}ss} \pi'(x')$, and $tls_i^r(\pi'(x')) = tls_i^1(\pi(x))$.

2. $\pi(x-1, Act) \in A_i^1$. Let a be the template action from which $\pi(x-1, Act)$ has been instantiated. Define $B_x = \pi(x)$ and $B'_x = \pi'(x')\pi'(x'+1)$, where $\pi'(x'-1) \to_{(a,1)} \pi'(x') \to_{(a,r)} \pi'(x'+1)$. The inductive hypothesis gives $\pi(x-1) \sim_{\overline{m}ss} \pi'(x'-1)$ and $tls_i^r(\pi'(x'-1)) = tls_i^1(\pi'(x'-1))$. Therefore, $\pi'(x'-1) \to_{(a,1)} \pi'(x') \to_{(a,r)} \pi'(x'+1)$ are valid transitions, $\pi(x) \sim_{\overline{m}ss} \pi'(x')$, $\pi(x) \sim_{\overline{m}ss} \pi'(x'+1)$, and $tls_i^r(\pi'(x'+1)) = tls_i^1(\pi'(x'))$.

Simulation requirement (iii) is therefore satisfied. It follows that $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{n}')$. By the inductive application of the latter and by transitivity of $\leq_{\overline{m}ss}$, we obtain that $\mathcal{S}(\overline{c}) \leq_{\overline{m}ss} \mathcal{S}(\overline{n})$.

□

Corollary 6.3. *Let $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V}) \in \mathbb{SFE}$ be a PIIS of k roles. Let $\forall_{\overline{v}}\phi(\overline{v})$ be an $\overline{m}$-indexed formula. Then, $\mathcal{S}(\overline{c}) \models \forall_{\overline{v}}\phi(\overline{v})$ iff $\forall \overline{n} \geq \overline{c}.\, \mathcal{S}(\overline{n}) \models \forall_{\overline{v}}\phi(\overline{v})$, where $\overline{c} = cutoff\_SFE(\mathcal{S}, \overline{m})$.*

Proof. Let $\overline{n} \geq \overline{c}$ be arbitrary. By Theorem 3.1 it suffices to show that $\mathcal{S}(\overline{n}) \leq_{\overline{m}ss} \mathcal{S}(\overline{c})$ and $\mathcal{S}(\overline{c}) \leq_{\overline{m}ss} \mathcal{S}(\overline{n})$. The former is given by by Lemma 6.7. The latter is given by Lemma 6.8. □

Semantics	$\sim_{\mathbf{aes}}$	Cutoff	Soundness	Completeness	#Systems to check
$\mathbb{SMR}$	Yes	$(\max(1, \bar{m}.i + \lvert D_i \rvert))_{1 \leq i \leq \lvert\mathcal{T}\rvert}$	Yes	No	$\prod_{1 \leq i \leq k} (\bar{c}(i) - \bar{m}(i) + 1)$
$\mathbb{SGS}$	Yes	$(\max(1, \bar{m}.i))_{1 \leq i \leq \lvert\mathcal{T}\rvert}$	Yes	No	1
$\mathbb{SFE}$	No	$(\max(1, \bar{m}.i))_{1 \leq i \leq \lvert\mathcal{T}\rvert}$	Yes	Yes	1

Table 6.1: Comparison of the $\mathbb{SMR}$, $\mathbb{SGS}$, and $\mathbb{SFE}$ classes.

This concludes the proof of Theorem 6.3. Corollary 6.3 gives a methodology for solving the PMCP by model checking the cutoff system. Differently from Corollary 6.1 and Corollary 6.2, where certain conditions are assumed, by means of Corollary 6.3 we always solve the PMCP for any given system and specification. In other words, while SMR and SGS do not provide an answer to the PMCP in the absence of an agent-environment simulation, SFE always provides a solution. Therefore, the PMCP for the $\mathbb{SFE}$ class of PIIS is decidable.

Corollary 6.4. *The parameterised model checking problem for the $\mathbb{SFE}$ class of PIIS is decidable.*

6.5 Conclusions

In this chapter we developed a methodology to solve the parameterised model checking problem for a number of noteworthy classes of PIIS. As we showed, when a cutoff can be determined, the parameterised model checking problem can be solved via standard model checking by verifying all system instances up to the cutoff.

Table 6.1 summarises the theoretical results obtained. Since the parameterised model checking problem is in general undecidable, no complete results can be established in general. In section 6.2 we presented an incomplete technique for the $\mathbb{SMR}$ class. In section 6.3 we analysed an incomplete technique for the $\mathbb{SGS}$ class. In section 6.4 we studied a complete technique for the $\mathbb{SFE}$ class. Incompleteness for the $\mathbb{SMR}$ and the $\mathbb{SGS}$ classes follows in the absence of an agent-environment simulation between the templates. By contrast, we can always assess the correctness of a specification on $\mathbb{SFE}$ systems. However, this comes with considerable limitations to the range of systems the technique can be applied to. For instance, the result cannot be applied to any scenario where the agents evolve in any other way other than lock-step evolution. Other systems may be modelled as $\mathbb{SMR}$ or $\mathbb{SGS}$ systems. $\mathbb{SMR}$ is suitable for scenarios requiring refined interactions between agents of different roles, whereas $\mathbb{SGS}$ is suitable for simulating synchronous semantics. In general, the technique for the former class returns larger cutoffs than those for the latter class. Both techniques are limited by the requirement of an agent-environment simulation between the agent and environment templates. This makes it difficult to model certain applications of interest, such as cache coherence protocols [BM04].

Chapter 7

MCMAS-P: A model checker for the verification of unbounded multiagent systems

In this chapter we introduce `MCMAS-P`, an experimental model checking toolkit that implements the techniques presented in chapters 5 and 6. `MCMAS-P` is implemented in `C++` and relies on the BDD-based checker `MCMAS` [LQR09, LQR15] for any check on the concrete models. In its current version, `MCMAS-P` supports indexed ACTLK$\backslash X$ formulae.

7.1 Implementation details

`MCMAS-P` takes PISPL descriptions as input. PISLP extends ISPL, the MCMAS input language, by closely following the framework of PIIS described in section 3.2. In particular, a PISPL file provides the agent and environment templates, their local states, their asynchronous, agent-environment, role-synchronous, global synchronous, and multi-role actions, their protocols, and their evolution functions. Figure 7.1 shows the PISPL encoding of the Train-Gate-Controller.

Figure 7.2 presents the key steps carried out by the checker. Given a PIIS $\mathcal{S} = (\mathcal{T}, \mathcal{E}, \mathcal{V})$

```
Template Environment
  Vars:
    state : {PG,NG,R};
  end Vars
  InitState
    state = PG;
  end InitState
  Protocol:
    state = PG : {n_lock , p_enter };
    state = NG  : {p_lock , n_enter };
    state = R  : {p_exit , n_exit };
  end Protocol
  Evolution:
    state = PG if Action = p_lock
                 or Action = p_exit;
    state = NG  if Action = n_lock
                 or Action = n_exit;
    state = R  if Action = p_enter
                 or Action = n_enter;
  end Evolution
end Template

Template PTrain
  Vars:
    state : {W,T,A,TL};
  end Vars
  InitState
    state = W;
  end InitState
  Actions
    Asynchronous = {p_approach};
    AgentEnvironment  = {p_enter , p_exit };
    GlobalSynchronous = {p_lock , n_lock };
  end Actions
  Protocol:
    state = W  : {p_enter };
    state = T  : {p_exit };
        state = A  : {p_approach , n_lock };
        state = TL : {p_lock };
  end Protocol
  Evolution
    state = W  if Action = p_approach;
    state = T  if Action = p_enter;
    state = A  if Action = p_exit
                      or Action = p_lock;
        state = TL if Action = n_lock;
  end Evolution
end Template

Template NTrain  ... end Template

Evaluation
     pt if PTrain.state = T;
     nt if NTrain.state = T;
end Evaluation

Formulae
  (PTrain:{u,v},Train:{x,y})
  AG(
  pt(u) -> K(PTrain(u),!pt(v) and !nt(x))
  and
  nt(x) -> K(NTrain(x),!nt(y) and !pt(u))
  );
end Formulae
```

Figure 7.1: The PISPL encoding of the Train-Gate-Controller.

specified in PISPL and a set Λ of $\overline{m}$-indexed ACTLK$\backslash X$ formulae, the steps 2-6 are performed automatically by the checker. In the following, we describe these steps.

- In step 2, the PISPL input file is parsed. The declarations of the agent templates and the environment template are stored in temporary structures to be used in the following steps.

- In step 3, the checker determines the appropriate procedure to be invoked. The `SFE` procedure is used to verify the subclass $\mathbb{SFE}$ of PIIS, generated by agent templates defined only on asynchronous and global-synchronous actions. The `SGS` procedure is used to verify the subclass $\mathbb{SGS}$ of PIIS, generated by agent templates defined only on asynchronous, agent-environment, and global-synchronous actions. The `SMR` procedure is used to verify the subclass $\mathbb{SMR}$ of PIIS, generated by agent templates defined only on asynchronous, agent-environment, and multi-role actions. Finally, the `PIIS` procedure can be used to verify the subclass of PIIS that can always succeed in globally synchronising, as described

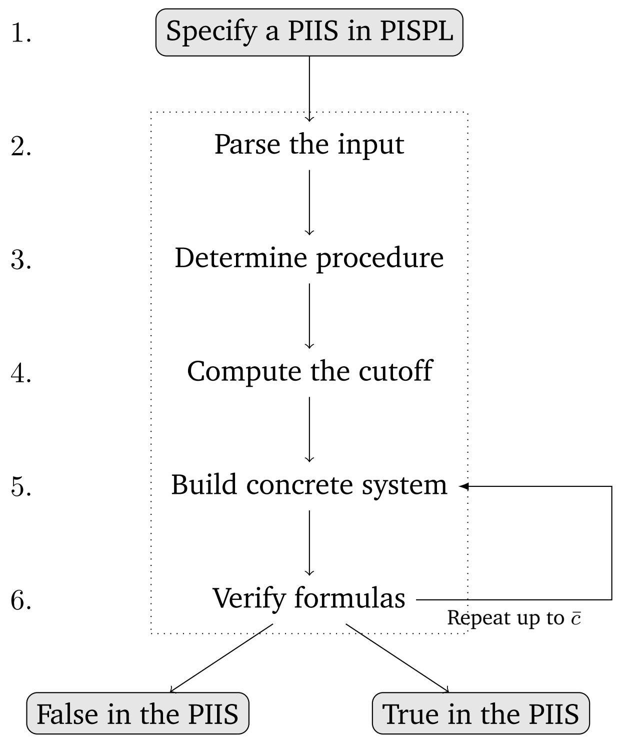


Figure 7.2: MCMAS-P architecture.

in chapter 5.

While the `SFE` procedure implements a complete cutoff technique, the other procedures implement incomplete cutoff techniques, since they insist on the compliance of certain conditions by the given PIIS. The condition of agent-environment simulation enforced by the `SGS` and `SMR` techniques is strictly stronger than the condition of gs-simulation enforced by the `PIIS` technique, i.e, every detectable cutoff by the `SGS` and `SMR` techniques is detectable by the `PIIS` technique. However, the `SGS` and `SMR` procedures terminate after performing a polynomial test whereas the `PIIS` procedure may never terminate.

Therefore, the checker first invokes the `SFE` procedure if $\mathcal{S} \in \mathbb{SFE}$. Otherwise, it performs the agent-environment simulation test if $\mathcal{S} \in \mathbb{SGS}$ or $\mathcal{S} \in \mathbb{SMR}$. With a successful simulation test, the `SGS` and `SMR` procedures are called, respectively, for the cutoff calculation. Otherwise, if either the agent-environment simulation test is not successful or $\mathcal{S} \notin \mathbb{SGS} \cup \mathbb{SMR}$, the `PIIS` procedure is initiated.

- In step 4, MCMAS-P calculates the cutoff $\bar{c}$ as in Algorithms 2, 3, 4, and 5. In the case of `PIIS`, the cutoff is identified by searching for a gs-simulation between a concrete system and the abstract system. The `SMR` procedure computes the cutoff from the action dependency sets and the cardinality of the sets of variables in the specifications to check. The SGS and SFE techniques calculate the cutoff only in terms of the latter.

- In step 5, the concrete system $\mathcal{S}(\overline{m})$ is built and encoded symbolically using the structures obtained in step 2. In step 6, the specification formulae Λ are reduced to their trivial instantiations $\Lambda[trivial]$ as in Lemma 3.1. MCMAS is then called to verify $\mathcal{S}(\overline{m})$ against $\Lambda[trivial]$. These steps are repeated for each concrete system up to the cutoff system $\mathcal{S}(\bar{c})$.

Following the above calculations the user can conclude whether or not a specification holds for any number of agents in the system. In the former case, all systems up to the cutoff system satisfy the specification and MCMAS-P returns *true*. In the latter case, at least one system up to the cutoff system does not satisfy the specification and MCMAS-P returns *false*.

7.1.1 Agent-environment simulation test

Instead of explicitly traversing the template transition relations, the agent-environment simulation test is more efficiently performed by utilising the OBDD representation of the templates. In particular, the test for an agent-environment simulation between the agent template $\mathcal{T}_i$ and the environment template is performed by checking the system composed of the two templates against a set of formulae expressing that whenever an AE_i action, or a GS action, or an MR_i action is enabled for the agent, the action is also enabled for the environment. This section describes this procedure.

The procedure is based on the assumption that the agent-environment actions, the global-synchronous actions, and the multi-role actions are enabled at exactly one state for the environment template. This allows us to check for an agent-environment simulation between an agent template $\mathcal{T}_i = (L_i, \iota_i, Act_i, P_i, t_i)$ and the environment template $\mathcal{E} = (L_\mathcal{E}, \iota_\mathcal{E}, Act_\mathcal{E}, P_\mathcal{E}, t_\mathcal{E})$ by model checking the interleaved interpreted system $\mathcal{S}_i$ against the set of formulae Δ_i, where $\mathcal{S}_i$ and Δ_i are defined as follows.

- $\mathcal{S}_i = (\mathcal{T}_i, \mathcal{E}, V_i)$ is the interleaved interpreted system composed of the agent template $\mathcal{T}_i$ and the environment template $\mathcal{E}$. The global states $G_i \subseteq L_i \times L_\mathcal{E}$ in $\mathcal{S}_i$ are assigned atomic propositions by the valuation function $V_i : L_i \times L_{\mathcal{E}} \to \mathcal{P}(AP)$, where $AP = \{a_i, a_{\mathcal{E}} \mid a \in AE_i \cup GS \cup MR_i\}$, defined as $a_i \in V_i((l, l_E))$ iff $a \in P_i(l)$ and $a_{\mathcal{E}} \in V_i((l, l_E))$ iff $a \in P_{\mathcal{E}}(l_E)$. In other words, a global state $g \in G_i$ is labelled with a_i ($a_{\mathcal{E}}$, respectively) if the action a is enabled for the agent template (the environment template, respectively) at g.

- $\Delta_i = \{AG(a_i \to a_{\mathcal{E}})\colon a \in AE_i \cup GS \cup MR_i\}$.

$\mathcal{S}_i$ satisfies the formulae in Δ_i iff there is an agent-environment simulation between $\mathcal{T}_i$ and $\mathcal{E}$.

Lemma 7.1. *$\mathcal{T}_i \leq_{aes} \mathcal{E}$ iff $\forall \delta \in \Delta_i . \mathcal{S}_i \models \delta$.*

Proof.

$\Rightarrow$ Suppose that $\mathcal{T}_i \leq_{aes} \mathcal{E}$. We show that $\forall \delta \in \Delta_i . \mathcal{S}_i \models \delta$. Let $\delta \in \Delta_i$. So, $\delta = AG(a_i \to a_{\mathcal{E}})$ for some $a \in AE_i \cup GS \cup MR_i$. Let π be an arbitrary path in $\mathcal{S}_i$. Suppose that $\pi(i) \models a_i$ for some $i \geq 1$. Since $\mathcal{T}_i \leq_{aes} \mathcal{E}$, there is an $i' \leq i$ with $\pi(i') \to_{A_i *} \pi(i)$ and $ls_i(\pi(i')) \sim_{aes} ls_{\mathcal{E}}(\pi(i'))$. Therefore, $a \in P_{\mathcal{E}}(\pi(i'))$, as otherwise we would have $ls_i(\pi(i')) \nsim_{aes} ls_{\mathcal{E}}(\pi(i'))$. So, $\pi(i') \models a_{\mathcal{E}}$, and therefore, $\pi(i) \models a_{\mathcal{E}}$. As i was arbitrary, $\pi \models AG(a_i \to a_{\mathcal{E}})$ follows. As π was arbitrary, $\mathcal{S}_i \models \delta$. Therefore, $\forall \delta \in \Delta_i . \mathcal{S}_i \models \delta$.

$\Leftarrow$ Suppose that $\forall \delta \in \Delta_i . \mathcal{S}_i \models \delta$. We show that $\mathcal{T}_i \leq_{aes} \mathcal{E}$. Let $\sim_{aes} = G_i$. We show that $\sim_{aes}$ is an agent-environment simulation between $\mathcal{T}_i$ and $\mathcal{E}$. Clearly, $\iota_i \sim_{aes} \iota_{\mathcal{E}}$. Let $g = (l, l_E) \in \sim_{aes}$ be arbitrary and suppose that $l \dashrightarrow_{A_i *} l^1 \dashrightarrow_a l^2$ for some $a \in AE_i \cup GS \cup MR_i$. We need to show that $l_E \dashrightarrow_a l_E^1$ for some l_E^1 with $(l^2, l_E^1) \in \sim_{aes}$. As l^1 is reachable from l through asynchronous actions, there is a global state g^1 reachable from g with $ls_i(g^1) = l^1$ and $ls_{\mathcal{E}}(g^1) = l_E$. Since $a \in P(l^1)$ and $\mathcal{S}_i \models AG(a_i \to a_{\mathcal{E}})$, $a \in P_{\mathcal{E}}(l_E)$ follows. As the action a is enabled at g^1, we obtain $(l^2, l_E^1) \in \sim_{aes}$ for $l_E^1 = t_{\mathcal{E}}(l_E, a)$. Therefore, $\mathcal{T}_i \leq_{aes} \mathcal{E}$ as required.

□

So, $\mathcal{T} \sim_{aes} \mathcal{E}$ iff $\mathcal{S}_i$ satisfies the formulae in Δ_i, for every agent template $\mathcal{T}_i$. This has considerable advantages in applications in terms of efficiency as the following section demonstrates.

Scenario	Procedure	Cutoff	Cutoff (s)	Reachable states	ACTLK$\backslash X$(s)
Train-Gate-Controller	SGS	(2,2)	0	64	0
Alpha algorithm	PIIS	3	507	177243	532
Robot foraging	SMR	(2,3)	0	648	0
Autonomous robot	SFE	(2,2)	0	15	0
MSI	PIIS	N/A	N/A	108	1
MESI	PIIS	N/A	N/A	173	2
MOESI	PIIS	N/A	N/A	216	6

Table 7.1: Verification results for parameterised model checking.

7.2 Experimental Results

This section reports experimental results obtained on the Train-Gate-Controller (section 3.2.1), the Alpha algorithm (section 5.3.2), the robot foraging scenario (section 3.2.1), the autonomous robots example (section 3.2.1), and the MSI, MESI, and MOESI cache coherence protocols [BM04]. The experiments were run on an Intel Core i7 CPU 3.4GHz with 8 GB RAM running Linux kernel version 3.19.4. The results are reported in Table 7.1 when verifying said scenarios against the following specifications, respectively.

$$\begin{aligned}
\phi_{TGC} &= \forall_{(\{u,v\},\{x,y\})} AG\left((pt,u) \rightarrow K^{u}_{ptrain}\left(\neg(pt,v) \wedge \neg(nt,x)\right)\right) \wedge \\
&\quad AG\left((nt,x) \rightarrow K^{x}_{ntrain}\left(\neg(nt,y) \wedge \neg(pt,u)\right)\right) \\
\phi_{AA} &= \forall_{\{v\}} K_v GF(con,v)) \\
\phi_{RFS} &= \forall_{(\{u\},\{x\})} AG\left((f,x) \rightarrow K^{u}_{TR}(f,x)\right) \\
\phi_{AR} &= \forall_{(\{v\},\{x\})} AG((h\_1,v) \rightarrow K^{v}_{TR1}(gr\_1,v) \wedge (h\_2,x) \rightarrow K^{x}_{TR2}(gr\_2,x)) \\
\phi_{cache} &= \forall_{\{i,j\}} AG((modified,i) \rightarrow K_i(invalid,j))
\end{aligned}$$

The cutoff detection time is reported in the fourth column, the number of reachable states of the cutoff model is shown in the fifth column, and the overall time required for the verification follows in the last column. For each scenario not reporting a cutoff, the corresponding specification was satisfied by the abstract model; thus the cutoff identification procedure did not take place. In these cases, the fifth column indicates the number of reachable states of the abstract model.

#Trains	#States	Time (s)	Memory (KiB)
(1,1)	16	0	**8774**
(10,10)	1.15×10^{27}	3	10005
(20,20)	2.30×10^{13}	128	47792
(30,30)	3.57×10^{19}	1317	60998
(40,40)	TIMEOUT	TIMEOUT	TIMEOUT

Table 7.2: Verification results for traditional model checking.

From the table, we can observe the time required by the PIIS procedure for the cutoff detection. This almost equals the time required for the whole verification process. This is generally expected since PIIS identifies a cutoff by building the abstract system which is of exponential size in the number of variables encoding the templates. In comparison, the table exemplifies the efficiency of the model checking procedures for the $\mathbb{SMR}$, $\mathbb{SGS}$, and $\mathbb{SFE}$ classes. These experimental results confirm advantages following the theoretical results when protocols can be expressed by these classes.

Finally, in comparison with the parameterised model checking techniques introduced in this thesis, Table 7.2 shows the intractability of the problems here considered in traditional model checking: the time and space requirements grow exponentially in the number of agents to consider. In our case the base model checker MCMAS could not verify the Train-Gate-Controller with 80 trains within the timeout of one hour.

Chapter 8

Conclusions

With the deployment of systems based on MAS-architectures there has been a growing interest in their verification. Considerable progress has been made in model checking MAS against specifications based on temporal, epistemic, deontic and strategic properties. Open-source implementations based on efficient symbolic approaches have been put forward and compared.

While this work has proven to be valuable, it is limited to scenarios where the number of components is known at design time. This is not a realistic assumption in certain MAS where the number of components cannot be known before deployment. A typical case is robotic swarms whereby the properties of the swarm need to hold irrespective of how many robots are present in the system.

8.1 Summary of thesis contributions

In this thesis we developed methodologies to solve the parameterised model checking problem for MAS in a number of semantical classes. Specifically, we introduced procedures for the cutoff identification of a given unbounded system. When a cutoff can be determined, the parameterised model checking problem can be solved by model checking all system instances up to the cutoff. We showed the procedures to be sound.

Table 8.1 summarises the theoretical results obtained.

Semantics	**Cutoff**	**Soundness**	**Completeness**	**#Systems to check**
PIS	$\dot{\mathcal{S}}(\overline{m}, td(\forall_{\overline{v}}\phi(\overline{v}) + 1)) \dot{\leq}_s \mathcal{S}(\overline{c})$	Yes	Yes	N/A
PIIS	$\hat{\mathcal{S}}(\overline{m}) \leq_{gs} \mathcal{S}(\overline{c})$	Yes	No	N/A
$\mathbb{SMR}$	$(\max(1, \overline{m}.i + \lvert D_i \rvert))_{1 \leq i \leq \lvert \mathcal{T} \rvert}$	Yes	No	$\prod_{1 \leq i \leq k} (\overline{c}(i) - \overline{m}(i) + 1)$
$\mathbb{SGS}$	$(\max(1, \overline{m}.i)_{1 \leq i \leq \lvert \mathcal{T} \rvert}$	Yes	No	1
$\mathbb{SFE}$	$(\max(1, \overline{m}.i))_{1 \leq i \leq \lvert \mathcal{T} \rvert}$	Yes	Yes	1

Table 8.1: Summary of theoretical results.

In chapter 4 we presented a sound and complete technique for parameterised interpreted systems. The cutoff detection procedure was based on the identification of a concrete system that can simulate the pruned computation forest of the abstract model. Soundness of the technique was assessed by means of cycle-stuttering simulations preserving the satisfaction of formulae up to a level of temporal depth. Upper bounds on the size of cutoffs were given from which the completeness of the procedure followed. The applicability of the procedure was theoretically showcased on the Beta swarm aggregation algorithm.

In chapter 5 we analysed a sound but incomplete technique for parameterised interleaved interpreted systems. Since the parameterised model checking problem for PIIS is in general undecidable, no complete results can be established in general. The cutoff detection procedure put forward relied on the existence of a gs-simulation between a concrete system and the abstract system. Intuitively, the condition of a gs-simulation expresses that all the agents can always succeed to globally synchronise. The application of the procedure on the Alpha algorithm and on cache coherence protocols illustrated that concrete scenarios often adhere to this condition.

Given the above procedure is in exponential space, in chapter 6 we identified the $\mathbb{SMR}$, $\mathbb{SGS}$, and $\mathbb{SFE}$ classes of PIIS for which we devised polynomial cutoff detection techniques. In section 6.2 we presented an incomplete technique for the $\mathbb{SMR}$ class. In section 6.3 we analysed an incomplete technique for the $\mathbb{SGS}$ class. In section 6.4 we studied a complete technique for the $\mathbb{SFE}$ class. Incompleteness for the $\mathbb{SMR}$ and the $\mathbb{SGS}$ classes follows in the absence of an agent-environment simulation between the templates, since in this case the techniques cannot assess the correctness of a given specification. By contrast, we can always assess the correctness of a specification on $\mathbb{SFE}$ systems. This level of confidence, which follows from the decidability result of Corollary 6.4, comes with considerable limitations to the range of systems the technique can be applied to. For instance, the result cannot be applied to any scenario where the agents evolve in any other way other than lock-step evolution. Other systems may be modelled as $\mathbb{SMR}$ or $\mathbb{SGS}$ systems. $\mathbb{SMR}$ is suitable for scenarios requiring refined interactions between agents of different roles, whereas $\mathbb{SGS}$ is suitable for simulating synchronous semantics. In general, the technique for the former class generally returns larger cutoffs than those for the latter class. Both techniques are limited by the requirement of an agent-environment simulation between the agent and environment templates. This makes it difficult to model certain applications of interest, such as cache coherence protocols.

The experiments obtained on the experimental model checker `MCMAS-P` that we introduced in chapter 7 confirm the correctness and attractiveness of the approach taken in this thesis.

8.2 Comparison with related work

We are not aware of previous work addressing parameterised verification for MAS. Traditional research in the verification of MAS involves the application of model checking techniques to systems with a well-defined number of participants. Therefore, these techniques can be used to verify certain instances of a given UMAS. However, since their time and space requirements are exponential in the number of agents, systems over a certain size are intractable to model check. As a result, no conclusions can be drawn on the correctness of the UMAS. Several existing approaches in model checking MAS were discussed in section 2.4.

Closer to the contribution presented in this thesis are studies on the parameterised verification of reactive systems. These were discussed in section 2.5. Differently from these works, where formalisms for reactive systems are considered, this thesis targeted the parameterised model checking problem for interpreted systems. As a result, the semantics presented in this thesis lead in general to different expressivity from the ones put forward for reactive systems. Still, combinations of the synchronisation primitives we introduced can simulate systems with broadcast labels [EN98], pairwise rendezvous labels [GS92], conjunctive and disjunctive guards [EK00]. However, even these cited research lines cater for temporal logics and no attempt is made to verify epistemic specifications.

Lastly, we note that the methodologies developed in this thesis are based on cutoffs. Existing cutoff techniques are not easily transferable to the context of this thesis. This is because of the branching nature of the knowledge modality that requires stronger notions of simulations than the ones used in reactive systems. Some notions of stuttering simulations previously defined in the context of CTL* [EN95, AJKR14] can be extended to epistemic logic. However, even these cited works cater for particular network topologies only and no attempt is made to identify the class of systems for which parameterised verification is decidable.

8.3 Future work

The verification of unbounded multiagent systems is an important and timely topic. This section identifies some possible directions of future work.

8.3.1 Dynamic UMAS and templates over unbounded variables

This thesis was concerned with the parameterised model checking problem defined as the problem of establishing whether a specification holds on a system comprised of an arbitrary number of agents. Often, however, in real-world scenarios, e.g., swarm robotics, agents may dynamically join or leave the system at runtime. This setting can be thought of as an adversarial setting where an adversary can modify the system at any time by adding or removing agents [NT15]. Clearly, a parameterised model checking technique, assessing the correctness of a protocol for any number of agents, cannot guarantee the correctness of a dynamic protocol. For instance, consider the Train-Gate-Controller (section 3.2.1) and a train that is removed from the system while the train is in the tunnel. Then, no train can ever enter the tunnel, since the controller (modelled by the environment) never switches the color of the traffic lights to green. Ideas from recent research in network protocols could be adapted to tackle this verification challenge. In particular, [NT15] argues that the coupling between participants in a dynamic system is likely to be weak, as is the case in the context of PIS and PIIS, since global invariants should be present irrespective of the adversarial actions. In such settings *composi-*

tional reasoning [1] has been shown effective, as also illustrated by [NT15] in verifying dynamic network protocols.

We also believe the techniques put forward in this thesis can serve as an ideal stepping stone to verify UMAS generated from templates over unbounded variables. While the templates here considered are finite structures, several protocols of interest, e.g., security [BCL09] and consensus [DLS88] protocols, include variables over unbounded domains. There are, therefore, two forms of unboundedness in these systems; one is the domain of the variables associated with each participant; the other is the number of participants. The associated parameterised model checking problem can be solved in two steps. The first step is the development of data abstraction techniques (see section 2.4.3), thereby abstracting the unbounded variables, and therefore obtaining finite-state system participants. The second step is the development of parameterised model checking techniques on the data abstracted systems.

8.3.2 Parameterised synthesis

Model checking is nowadays a mature approach to verification with increasingly sophisticated algorithms being applied to industrial settings. In these settings model checking tools are essentially used as debugging tools in that significant resources are already accommodated in developing a possibly erroneous program before model checking commences. To ameliorate costs concomitant with erroneous designs, *program synthesis* has been put forward. The *program synthesis problem* concerns the automatic generation of a program that is guaranteed to be correct w.r.t. a given specification. The problem is typically distinguished between *open* and *closed systems*. Differently from closed systems, in open systems the program interacts with the environment; a correct program should be able to handle any sequence of actions performed by the environment. Program synthesis for open systems is therefore naturally seen as a two-player game between the program and the environment, where the environment wins if the specification is violated, whereas the program wins if the specification is satisfied. The interested reader is referred to [MW84, Var96, VDMV98] and references therein for more details.

[1] Compositional reasoning techniques analyse each component of the system in isolation and infer global properties about the entire system (see [BCC98] and references therein).

The *parameterised synthesis problem* is the synthesis problem for parameterised systems. That is, given a specification, the parameterised synthesis problem concerns the generation of a parameterised system that is guaranteed to be correct w.r.t. the specification for any number of participants in the system. The problem is formalised for token rings in [JB12] where it is reduced to the synthesis problem with a fixed number of participants; the reduction is based on the cutoff results established for token rings in [EN95]. Then, bounded synthesis is adapted from [SF07] to solve the synthesis problem with a fixed number of participants. It is argued that any parameterised result based on static cutoffs (see section 2.5.2) can similarly be used to solve the parameterised synthesis problem. This is of particular interest in the context of the $\mathbb{SMR}$, $\mathbb{SGS}$, and $\mathbb{SFE}$ classes of PIIS which enjoy static cutoffs. In particular, in future work we aim to (i) reduce the parameterised synthesis problem for these classes to one of a bounded number of agents; and (ii) solve the synthesis problem on the cutoff systems.

8.3.3 Fault-tolerant UMAS

With the development of fault-tolerant MAS architectures [MSBG01, GFB05, KC00], which allow for a system's resilience to agents' faults, *fault injection* methodologies have been put forward for their verification. These inject faults on systems with correct behaviour and use model checking to verify a system's tolerance to the faults [EL09]. In future work we aim to devise automated methodologies for the verification of fault-tolerant UMAS. In particular, we will develop and implement a fault injection mechanism. More specifically, the agents' behavioural templates will be mutated to adhere a number of possible faults. The mutated templates will then be composed with the correct ones. Following this, the techniques here presented will be extended to assess a system's degree (the number, or the percentage, of faulty agents) of resilience before a property is violated. For instance, a robot operating a swarm robotics pattern formation protocol and broadcasting incorrect spatial positions may initiate the violation of the pattern formation the swarm has established. Finally, a taxonomy of specifications will be introduced, modelling diagnosability and recovery from faults.

Bibliography

[ACJT96] A. Abdulla, K. Cerans, B. Jonsson, and Y-K. Tsay. General decidability theorems for infinite-state systems. In *Proceedings of 11th Annual IEEE Symbosium on Logic in Computer Science (LICS96)*, pages 313–321. IEEE, 1996.

[AdAH$^+$00] R. Alur, L. de Alfaro, T. Henzinger, S. Krishnan, F. Mang, S. Qadeer, S. Rajamani, and S. Tasiran. MOCHA user manual. Technical report, University of California at Berkeley, 2000.

[ADHR07] P. Abdulla, G. Delzanno, N. Henda, and A. Rezine. Regular model checking without transducers (on efficient verification of parameterized systems). In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *Lecture Notes in Computer Science*, pages 721–736. Springer, 2007.

[AHH13] P. A. Abdulla, F. Haziza, and L. Holík. All for the price of few. In *Proceedings of the 14th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI13)*, volume 7737 of *Lecture Notes in Computer Science*, pages 476–495. Springer, 2013.

[AHK02] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

[AJ99] P. A. Abdulla and B. Jonsson. On the existence of network invariants for verifying parameterized systems. In *Correct System Design*, volume 1710 of *Lecture Notes in Computer Science*, pages 180–197. Springer, 1999.

[AJKR14] B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *Proceedings of the 15th International Conference on*

Verification, Model Checking, and Abstract Interpretation (VMCAI14), volume 8318 of *Lecture Notes in Computer Science*, pages 262–281. Springer, 2014.

[AJMd02] P. Abdulla, B. Jonsson, P. Mahata, and J. dâĂŹOrso. Regular tree model checking. In *Computer Aided Verification*, pages 555–568. Springer, 2002.

[AK86] K.R. Apt and D. C. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22(6):307–309, 1986.

[AKR$^+$14] B. Aminof, T. Kotek, S. Rubin, F. Spegni, and H. Veith. Parameterized model checking of rendezvous systems. In *CONCUR 2014–Concurrency Theory*, pages 109–124. Springer, 2014.

[APR$^+$01] T. Arons, A. Pnueli, S. Ruah, Y. Xu, and L. Zuck. Parameterized verification with automatically computed inductive assertions? In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV11)*, volume 6806 of *Lecture Notes in Computer Science*, pages 221–234. Springer, 2001.

[BAPM83] M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Acta Informatica*, 20(3):207–226, 1983.

[BCC98] Sergey Berezin, Sérgio Campos, and Edmund M Clarke. *Compositional reasoning in model checking*. Springer, 1998.

[BCCZ99] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS99)*, volume 1579 of *LNCS*, pages 193–207. Springer-Verlag, 1999.

[BCG88] M. C. Browne, E. M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59(12):115–131, 1988.

[BCK08] P. Baldan, A. Corradini, and B. KÃűnig. A framework for the verification of infinite-state graph transformation systems. *Information and Computation*, 206(7):869 – 907, 2008.

[BCL09] I. Boureanu, M. Cohen, and A. Lomuscio. A compilation method for the verification of temporal-epistemic properties of cryptographic protocols. *Journal of Applied Non-Classical Logics*, 19(4):463–487, 2009.

[BDT99] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence*. Oxford University Press, 1999.

[BFBD13] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013.

[BFVW06] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifying multi-agent programs by model checking. *Autonomous Agents and Multi-Agent Systems*, 12(2):239–256, 2006.

[BJ$^+$00] A. Bouajjani, , B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *Computer Aided Verification*, Lecture Notes in Computer Science, pages 403–418. Springer, 2000.

[BLS02] K. Baukus, Y. Lakhnech, and K. Stahl. Parameterized verification of a cache coherence protocol: Safety and liveness. In *Verification, Model Checking, and Abstract Interpretation*, Lecture Notes in Computer Science, pages 317–330. Springer, 2002.

[BM04] K. Baukus and R. Meyden. A knowledge based analysis of cache coherence. In *Formal Methods and Software Engineering*, pages 99–114. Springer, 2004.

[Bry92] R. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3):293–318, 1992.

[CDLQ09a] M. Cohen, M. Dam, A. Lomuscio, and H. Qu. A data symmetry reduction technique for temporal-epistemic logic. In *Proceedings of the 7th International Symposium on Automated Technology for Verification and Analysis (ATVA 09)*, volume 5799 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2009.

[CDLQ09b] M. Cohen, M. Dam, A. Lomuscio, and H. Qu. A symmetry reduction technique for model checking temporal-epistemic logic. In *Proceedings of the 21st International*

Joint Conference on Artificial Intelligence (IJCAI09), pages 721–726, Pasadena, USA, 2009.

[CDLR09] M. Cohen, M. Dam, A. Lomuscio, and F. Russo. Abstraction in model checking multi-agent systems. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS09)*, pages 945–952. IFAAMAS Press, 2009.

[CE81] E. M. Clarke and E. Emerson. Design and synthesis of synchronization skeletons for branching-time temporal logic. In *Proceedings of Workshop on Logic of Programs*, volume 131 of *LNCS*, pages 52–71. Springer-Verlag, 1981.

[CG87] E. Clarke and O. Grumberg. Avoiding the state explosion problem in temporal logic model checking. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 294–303. ACM, 1987.

[CGB89] E.M. Clarke, O. Grumberg, and M.C. Browne. Reasoning about networks with many identical finite state processes. *Information and Computation*, 81(1):13–31, 1989.

[CGL94] E. M. Clarke, O. Grumberg, and D. Long. Model checking and abstractions. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[CLMM14] P. Cermák, A. Lomuscio, F. Mogavero, and A. Murano. MCMAS-SLK: A model checker for the verification of strategy logic specifications. In *Proceedings of the 26th International Conference on Computer Aided Verification (CAV14)*, volume 8559 of *Lecture Notes in Computer Science*, pages 525–532. Springer, 2014.

[CLMM15] P. Cermák, A. Lomuscio, F. Mogavero, and A. Murano. Verifying and synthesising multi-agent systems against one-goal strategy logic specifications. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI15)*, pages 2038–2044. AAAI Press, 2015.

[CTTV04] E. Clarke, M. Talupur, T. Touili, and H. Veith. Verification by network decomposition. In *Proceedings of the 15th International Conference on Concurrency Theory (CONCUR04)*, volume 3170 of *Lecture Notes in Computer Science*, pages 276–291. Springer, 2004.

[CTV06] E. Clarke, M. Talupur, and H. Veith. Environment abstraction for parameterized verification. In *Proceeding of the 7th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI06)*, pages 126–141. Springer, 2006.

[CTV08] E.M. Clarke, M. Talupur, and H. Veith. Proving ptolemy right: The environment abstraction framework for model checking concurrent systems. In *Proceedings of 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS08)*, volume 4963 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2008.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[DRB02] G. Delzanno, , J. Raskin, and L. Van Begin. Towards the automated verification of multithreaded java programs. In *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 173–187. Springer, 2002.

[DSZ10] G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized verification of ad hoc networks. In *Proceedings of the 21'st International Conference on Concurrency Theory (CONCUR10)*, volume 6269 of *Lecture Notes in Computer Science*, pages 313–327. Springer, 2010.

[DSZ11] G. Delzanno, A. Sangnier, and G. Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *Foundations of Software Science and Computational Structures*, pages 441–455. Springer, 2011.

[DT98] C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 313–329. Springer, 1998.

[DWFZ12] Clare Dixon, Alan FT Winfield, Michael Fisher, and Chengxiu Zeng. Towards temporal verification of swarm robotic systems. *Robotics and Autonomous Systems*, 60(11):1429–1441, 2012.

[EFM99] J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Proceedings of the 10th International Symposium on Logic in Computer Science (LICS99)*, pages 352–359. IEEE, 1999.

[EK00] E. Emerson and V. Kahlon. Reducing model checking of the many to the few. In *Proceedings of the 17th International Conference on Automated Deduction (CADE00)*, volume 1831 of *Lecture Notes in Computer Science*, pages 236–254. Springer, 2000.

[EK02] E. A. Emerson and V. Kahlon. Model checking large-scale and parameterized resource allocation systems. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS08)*, volume 2280 of *Lecture Notes in Computer Science*, pages 251–265. Springer, 2002.

[EK03a] A. Emerson and V. Kahlon. Exact and efficient verification of parameterized cache coherence protocols. In *Correct Hardware Design and Verification Methods*, pages 247–262. Springer, 2003.

[EK03b] E.A. Emerson and V. Kahlon. Model checking guarded protocols. In *Proceedings of the 14th International Symposium on Logic in Computer Science (LICS03)*, pages 361–370. IEEE, 2003.

[EK04] A. Emerson and V. Kahlon. Parameterized model checking of ring-based message passing systems. In *Computer Science Logic*, pages 325–339. Springer, 2004.

[EL09] J. Ezekiel and A. Lomuscio. 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, 2009.

[ELMV11] J. Ezekiel, A. Lomuscio, L. Molnar, and S. Veres. 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, 2011.

[Eme90] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, pages 996–1071. Elsevier Science Publishers, 1990.

[EN95] E.A. Emerson and K.S. Namjoshi. Reasoning about rings. In *Proceedings of the 22nd Annual Sigact-Aigplan on Principles of Programming Languages (POPL95)*, pages 85–94. Pearson Education, 1995.

[EN96] E. A. Emerson and K. S. Namjoshi. Automatic verification of parameterized synchronous systems. In *Proceedings of the 8th International Conference one Computer Aided Verification (CAV96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 87–98. Springer, 1996.

[EN98] E. A. Emerson and K. S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *Proceedings of 13th International Symposium on Logic in Computer Science (LICS98)*, pages 70–80. IEEE, 1998.

[Eng06] Andries P Engelbrecht. *Fundamentals of computational swarm intelligence*. John Wiley & Sons, 2006.

[ES96] E.A. Emerson and A.P. Sistla. Symmetry and model checking. *Formal methods in system design*, 9(1):105–131, 1996.

[FHMV95] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, 1995.

[FHV95] R. Fagin, J. Y. Halpern, and M. Vardi. A nonstandard approach to the logical omniscience problem. *Artificial Intelligence*, 79, 1995.

[GFB05] Zahia Guessoum, Nora Faci, and Jean-Pierre Briot. Adaptive replication of large-scale multi-agent systems–towards a fault-tolerant multi-agent platform. In *Software Engineering for Multi-Agent Systems IV*, pages 238–253. Springer, 2005.

[GS92] S. M. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992.

[GvdM04] P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *Proceedings of 16th International Conference on Computer Aided Verification (CAV04)*, volume 3114 of *LNCS*, pages 479–483. Springer-Verlag, 2004.

[HBR09] Y. Hanna, S. Basu, and H. Rajan. Behavioral automata composition for automatic topology independent verification of parameterized systems. In *Proceedings of the 7th Joing Meeting of the European Software Engineering Conference and the ACM Sigsoft Symposium on the Foundations of Software Engineering (ESEC/FSE09)*, pages 325–334. ACM, 2009.

[HR00] M. R. A. Huth and M. D. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, Cambridge, England, 2000.

[HSBR10] Y. Hanna, D. Samuelson, S. Basu, and H. Rajan. Automating cut-off for multi-parameterized systems. In *Proceedings of the 12th International Conference on Formal Engineering Methods (ICFEM10)*, volume 6447 of *Lecture Notes in Computer Science*, pages 338–354. Springer, 2010.

[HW02a] W. Hoek and M. Wooldridge. Model checking knowledge and time. In *SPIN 2002 – Proceedings of the Ninth International SPIN Workshop on Model Checking of Software*, 2002.

[HW02b] W. van der Hoek and M. Wooldridge. Tractable multiagent planning for epistemic goals. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS02)*, pages 1167–1174. ACM Press, 2002.

[JB12] Swen Jacobs and Roderick Bloem. Parameterized synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 362–376. Springer, 2012.

[KC00] Sanjeev Kumar and Philip R Cohen. Towards a fault-tolerant multi-agent system architecture. In *Proceedings of the fourth international conference on Autonomous agents*, pages 459–466. ACM, 2000.

[KKW10] A. Kaiser, D. Kroening, and T. Wahl. 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, 2010.

[KL13a] P. Kouvaros and A. Lomuscio. Automatic verification of parameterised interleaved multi-agent systems. In *Proceedings of the 12th International Conference on Autonomous Agents and Multi-Agent systems (AAMAS13)*, pages 861–868. IFAAMAS, 2013.

[KL13b] P. Kouvaros and A. Lomuscio. A cutoff technique for the verification of parameterised interpreted systems with parameterised environments. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI13)*, pages 2013–2019. AAAI Press, 2013.

[KL15a] P. Kouvaros and A. Lomuscio. A counter abstraction technique for the verification of robot swarms. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI15)*, pages 2081–2088. AAAI Press, 2015.

[KL15b] P. Kouvaros and A. Lomuscio. Verifying emergent properties of swarms. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI15)*, pages 1083–1089. AAAI Press, 2015.

[KL16] P. Kouvaros and A. Lomuscio. Parameterised verification for multi-agent systems. *Artificial Intelligence*, 234:152–189, 2016.

[KLN$^+$06] M. Kacprzak, A. Lomuscio, A. Niewiadomski, W. Penczek, F. Raimondi, and M. Szreter. Comparing BDD and SAT based techniques for model checking Chaum's dining cryptographers protocol. *Fundamenta Informaticae*, 63(2,3):221–240, 2006.

[KLP04] M. Kacprzak, A. Lomuscio, and W. Penczek. Verification of multiagent systems via unbounded model checking. In *Proceedings of the Third International Conference on Autonomous Agents and Multiagent Systems (AAMAS04)*, pages 638–645. ACM, 2004.

[KM69] R. Karp and R. Miller. Parallel program schemata. *Journal of Computer and system Sciences*, 3(2):147–195, 1969.

[KM89] R. Kurshan and K. McMillan. A structural induction theorem for processes. In *Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, pages 239–247. ACM, 1989.

[KNN$^+$08] M. Kacprzak, W. Nabialek, A. Niewiadomski, W. Penczek, A. Pólrola, M. Szreter, B. Woźna, and A. Zbrzezny. Verics 2007 - a model checker for knowledge and real-time. *Fundamenta Informaticae*, 85(1):313–328, 2008.

[KP04a] M. Kacprzak and W. Penczek. A SAT-based approach to unbounded model checking for alternating-time temporal epistemic logic. *Synthese*, 142(2):203–227, 2004.

[KP04b] M. Kacprzak and W. Penczek. Unbounded model checking for alternating-time temporal logic. In *Proceedings of the 3rd International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS04)*, volume II, pages 646–653. ACM, 2004.

[Liu07] *Strategies for energy optimisation in a swarm of foraging robots*, volume 4433 of *Lecture Notes in Computer Science*. Springer, 2007.

[LŁP03] A. Lomuscio, T. Łasica, and W. Penczek. Bounded model checking for interpreted systems: preliminary experimental results. In *Proceedings of the 2nd International Workshop on Formal Approaches to Agent-Based Systems (FAABS04)*, volume 2699 of *Lecture Notes in Computer Science*, pages 115–125. Springer, 2003.

[LPQ10] A. Lomuscio, W. Penczek, and H. Qu. Partial order reduction for model checking interleaved multi-agent systems. *Fundamenta Informaticae*, 101(1–2):71–90, 2010.

[LPW07] A. Lomuscio, W. Penczek, and B. Woźna. Bounded model checking knowledge and real time. *Artificial Intelligence*, 171(16-17):1011–1038, 2007.

[LQR09] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *Proceedings of the 21th International Conference on Computer Aided Verification (CAV09)*, volume 5643 of *Lecture Notes in Computer Science*, pages 682–688. Springer, 2009.

[LQR15] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. *Software Tools for Technology Transfer*, 2015. To Appear.

[LQS11] A. Lomuscio, H. Qu, and M. Solanki. Towards verifying contract regulated service composition. *Journal of Autonomous Agents and Multi-Agent Systems*, 24(3):345–373, 2011.

[LR06a] A. Lomuscio and F. Raimondi. The complexity of model checking concurrent programs against CTLK specifications. In *DALT*, volume 4327 of *Lecture Notes in Computer Science*, pages 29–42. Springer, 2006.

[LR06b] A. Lomuscio and F. Raimondi. Model checking knowledge, strategies, and games in multi-agent systems. In *Proceedings of the 5th International Joint Conference on Autonomous agents and Multi-Agent Systems (AAMAS06)*, pages 161–168. ACM Press, 2006.

[LS03] A. Lomuscio and M. Sergot. Deontic interpreted systems. *Studia Logica*, 75(1):63–92, 2003.

[Mai01] M. Maidl. A unifying model checking approach for safety properties of parameterized systems. In *Computer Aided Verification*, Lecture Notes in Computer Science, pages 311–323. Springer, 2001.

[McM02] K. L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV02)*, volume 2404 of *LNCS*, pages 250–264. Springer-Verlag, 2002.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer-Verlag, 1980.

[MS99] R. van der Meyden and H. Shilov. Model checking knowledge and time in systems with perfect recall. In *Proceedings of the 19th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS99)*, volume 1738 of *Lecture Notes in Computer Science*, pages 432–445. Springer, 1999.

[MS04] E. M. Maximilien and M. P. Singh. A framework and ontology for dynamic web services selection. *Internet Computing*, 8(5):84–93, 2004.

[MSBG01] Olivier Marin, Pierre Sens, Jean-Pierre Briot, and Zahia Guessoum. Towards adaptive fault tolerance for distributed multi-agent systems. In *Proceedings of ERSADS*, pages 195–201, 2001.

[Mur00] R. R. Murphy. Marsupial and shape-shifting robots for urban search and rescue. *Intelligent Systems and their Applications*, 15(2):14–19, 2000.

[MW84] Zohar Manna and Pierre Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 6(1):68–93, 1984.

[Nem05] J. Nembrini. *Minimalist Coherent Swarming of Wireless Networked Autonomous Mobile Robots*. PhD thesis, University of the West of England, 2005.

[NT15] Kedar S Namjoshi and Richard J Trefler. Analysis of dynamic process networks. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 164–178. Springer, 2015.

[Pel93] D. Peled. All from one, one for all: on model checking using representatives. In *Proceedings of the 5th International Conference on Computer Aided Verification (CAV93)*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer, 1993.

[PL03] W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multi-agent systems (AAMAS03)*, pages 209–216. IFAAMAS, 2003.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th International Symposium Foundations of Computer Science (FOCS77)*, pages 46–57, 1977.

[PS00] A. Pnueli and E. Shahar. Liveness and acceleration in parameterized verification. In *Proceedings of the 12th International Conference on Computer Aided Verifica-*

tion (CAV00), volume 1855 of *Lecture Notes in Computer Science*, pages 328–343. Springer, 2000.

[PWZ02] W. Penczek, B. Woźna, and A. Zbrzezny. Bounded model checking for the universal fragment of CTL. *Fundamenta Informaticae*, 51(1-2):135–156, 2002.

[PXZ02] A. Pnueli, J. Xu, and L. Zuck. 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, 2002.

[Rai06] F. Raimondi. *Model Checking Multi-Agent Systems*. PhD thesis, 2006.

[RL04] F. Raimondi and A. Lomuscio. Automatic verification of deontic interpreted systems by model checking via OBDDs. In *Proceedings of the Sixteenth European Conference on Artificial Intelligence (ECAI04)*, pages 53–57. IOS PRESS, 2004.

[RL05] F. Raimondi and A. Lomuscio. Automatic verification of multi-agent systems by model checking via OBDDs. *Journal of Applied Logic*, 5(2):235–251, 2005.

[RZSH$^+$14] A. Rosenfeld, I. Zuckerman, E. Segal-Halevi, O. Drein, and S. Kraus. Negochat: a chat-based negotiation agent. In *Proceedings of the 14th International Conference on Autonomous Agents and Multi-Agent systems (AAMAS14)*, pages 525–532. IFAAMAS, 2014.

[Şah05] Erol Şahin. Swarm robotics: From sources of inspiration to domains of application. In *Swarm robotics*, pages 10–20. Springer, 2005.

[SF07] Sven Schewe and Bernd Finkbeiner. Bounded synthesis. In *Automated Technology for Verification and Analysis*, pages 474–488. Springer, 2007.

[SG90] Z. Shtadler and O. Grumberg. Network grammars, communication behaviors and automatic verification. In *Automatic Verification Methods for Finite State Systems*, Lecture Notes in Computer Science, pages 151–165. Springer, 1990.

[SLB08] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

[Som05] F. Somenzi. CUDD: CU decision diagram package - release 2.4.0. http://vlsi.colorado.edu/ fabio/CUDD/cuddIntro.html, 2005.

[ŞW08] Erol Şahin and Alan Winfield. Special issue on swarm robotics. *Swarm Intelligence*, 2(2):69–72, 2008.

[SWJ08] M. Saksena, O. Wibling, and B. Jonsson. Graph grammar modeling and verification of ad hoc routing protocols. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pages 18–32. Springer, 2008.

[Szy88] B. K. Szymanski. A simple solution to lamport's concurrent programming problem with linear wait. In *Proceedings of the 2nd International Conference on Supercomputing (ICS88)*, pages 621–626. ACM, 1988.

[Var96] Moshe Y Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for concurrency*, pages 238–266. Springer, 1996.

[vdMS04] R. van der Meyden and K. Su. Symbolic model checking the knowledge of the dining cryptographers. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW04)*, pages 280–291. IEEE Computer Society, 2004.

[VDMV98] Ron Van Der Meyden and Moshe Y Vardi. Synthesis from knowledge-based specifications. In *CONCUR'98 Concurrency Theory*, pages 34–49. Springer, 1998.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the 1st Symposium on Logic in Computer Science*, pages 332–344, Cambridge, 1986.

[Wei99] G. Weiss. *Multi-agent systems*. MIT Press, 1999.

[WL90] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems (AVMFSS89)*, volume 407 of *Lecture Notes in Computer Science*, pages 68–80. Springer, 1990.

[WLNM08] A. Winfield, W. Liu, J. Nembrini, and A. Martinoli. Modelling a wireless connected swarm of mobile robots. *Swarm Intelligence*, 2(2-4):241–266, 2008.

[WLP05] B. Wozna, A. Lomuscio, and W. Penczek. Bounded model checking for deontic interpreted systems. In *Proceedings of the 2nd International Workshop on Logic and communication in Multi-Agent Systems*, pages 93–114. Elsevier, 2005.

[Woo09] M. Wooldridge. *An introduction to MultiAgent systems*. Wiley, second edition edition, 2009.

[YL10] Q. Yang and M. Li. A cut-off approach for bounded verification of parameterized systems. In *Proceedings of the 32nd International Conference on Software Engineering (ICSE10)*, pages 345–354. IEEE, 2010.

[ZP04] L. Zuck and A. Pnueli. Model checking and abstraction to the aid of parameterized systems (a survey). *Computer Languages, Systems & Structures*, 30(3):139–169, 2004.