Parameterised Verification of Randomised Distributed Systems using State-based Models

by

Douglas Graham

A thesis submitted to the
Faculty of Information and Mathematical Sciences
at the University of Glasgow
for the degree of
Doctor of Philosophy

February 2008

Abstract

Model checking is a powerful technique for the verification of distributed systems but is limited to verifying systems with a fixed number of processes. The verification of a system for an arbitrary number of processes is known as the parameterised model checking problem and is, in general, undecidable. Parameterised model checking has been studied in depth for non-probabilistic distributed systems. We extend some of this work in order to tackle the parameterised model checking problem for distributed protocols that exhibit *probabilistic* behaviour, a problem that has not been widely addressed to date.

In particular, we consider the application of *network invariants* and explicit *induction* to the parameterised verification of state-based models of randomised distributed systems. We demonstrate the use of network invariants by constructing invariant models for non-probabilistic and probabilistic forms of a simple counter token ring protocol. We show that proving properties of the invariants equates to proving properties of the token ring protocol for any number of processes.

The use of induction is considered for the verification of a class of randomised distributed systems. These systems, termed *degenerative*, have the property that a model of a system with given communication graph eventually behaves like a model of a system with a *reduced* graph, where reduction is by removal of a *set* of nodes. We distinguish between *deterministically*, *probabilistically* and *semi*-degenerative systems, according to the manner in which a system degenerates. For the former two classes we describe induction schemas for reasoning about models of these systems over arbitrary communication graphs. We show that certain properties hold for models of such systems with any graph if they hold for all models of a system with some *base* graph and demonstrate this via case studies: two randomised leader election protocols. We illustrate how induction can also be employed to prove properties of semi-degenerative systems by considering a simple gossip protocol.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisor, Muffy Calder, for her help and guidance. Her support and encouragement throughout the course of this PhD were invaluable. Thanks also go to my second supervisor, Alice Miller, who had a considerable amount of input into this work and who was always available to give assistance when required.

I would also like to acknowledge Paolo Ballarini, Alastair Donaldson, Simon Gay, Rob Irving and Gethin Norman for their assistance.

My research was supported by a University of Glasgow 2004 Postgraduate Research Scholarship.

Finally, special thanks must go to my fiancée and my family for their patience and support.

Chapter 1

Introduction

Outline In this chapter we provide some context for the thesis, introducing *model checking* and *probabilistic model checking*. We define the *parameterised model checking problem* in particular and highlight some of the work that has been carried out on this to date. We then describe our work on the problem, providing an outline of the thesis whilst highlighting our contribution to the body of work before concluding with a thesis statement.

1.1 Verification of Distributed Systems

Software and hardware systems are being used more and more for safety-critical applications and hence reliability has become a key issue in the systems development process. This is especially true of *distributed* (or *concurrent*) systems, consisting of a set of *processes* executing in parallel that can communicate by some means (for example by a set of shared variables or over a set of communication channels). The interleaving of process executions makes analysis of distributed systems complex and therefore such systems are difficult to guarantee as reliable.

A common means of ensuring reliability of a system is by *validation*, whereby a set of test-cases is input to a system and the output generated by the system is checked against the expected output. However, it is difficult to produce a set of cases that will guarantee complete coverage of a system, particularly if it is of a complex nature as in the case of distributed systems. An alternative to validation is *verification*, whereby a system is described formally and the description is verified against some specification. Proving correctness increases confidence in the reliability of the system.

Theorem proving and *model checking* are two commonly used techniques for verification of distributed systems. Theorem provers, such as PVS [51], can be used to automate parts of a verification proof. Although a useful verification technique, theorem proving requires a considerable amount of effort in order to establish a theoretical framework for a particular proof. We do not consider theorem proving further.

Model checking is a verification technique whereby a model (generally defined in terms of a state-transition system) of a system is defined according to some system specification. Properties that capture the correct behaviour of the system are formally defined (for example using a temporal logic) and verified by an automated search of the full state space of the model. Model checking is usually employed for verification of *finite state* systems but it is also possible to verify infinite-state systems using this technique. We focus on verifying finite state systems.

Formally, the *model checking problem* is to establish, for a model $\mathcal{M}$ and a property ϕ, whether $\mathcal{M} \models \phi$ ($\mathcal{M}$ satisfies ϕ). Several formalisms for specifying $\mathcal{M}$ and ϕ exist. When reasoning about concurrent software/hardware systems we often employ a *temporal logic* to formalise ϕ, and $\mathcal{M}$ is usually described by some form of finite-state automata model (e.g. a Kripke structure).

A variety of algorithms exist to check that a temporal logic property is true in some model. These algorithms have been employed in a number of different model checking tools. These tools allow the user to write the specification of a system using a modelling language and to define properties in some temporal logic. The model checker will then build an internal representation of the model using the system specification and search it to verify a property, giving back a true/false answer and possibly a counter example if the property does not hold. Two examples of such model checking tools are SMV [47] and SPIN [32], which use different modelling languages, underlying verification algorithms and temporal logics, but serve the same purpose.

1.2 Verification of Randomised Distributed Systems

One of the key features of traditional model checking is non-determinism. This has proven to be an essential element in traditional model checking tools since it allows details of a system (or its implementation) to be abstracted away in the model, and it provides the means to model the interleavings of concurrent processes executions arbitrarily. In order to verify a model with non-determinism, a model checking tool must explore all *possible* non-deterministic choices. However, there are some instances under which it is not sufficient to simply examine the possible choices but it is also necessary to look at the *likelihood* of making a choice. This is the basis of *probabilistic model checking.*

Probabilistic model checking has become an important area of research due to the increased use of probabilistic algorithms and the requirement for analysis of not just system correctness but also system performance. Probabilistic model checkers, such as PRISM [40], Rapture [36] and LiQuor [7], enable properties such as *"the system will fail with probability less than 0.01"* and *"with probability 1, a leader will be elected"* to be verified.

Probabilistic model checking is similar in many ways to traditional model checking. Indeed, the model checking problem can be stated in exactly the same way for both. The main difference, then, lies in the representation of the model of a system, and the logic that is used to describe a system's properties. As for traditional model checking, probabilistic verification involves traversing the state space of a model to determine reachable states, but in addition to this the probability of reaching these states must be calculated.

To understand the benefits of probabilistic model checking, consider the issues of modelling failure and analysing system performance. In the case of the former, although it is possible to capture the concept of erroneous behaviour using traditional models, in some cases it may not be enough just to consider whether an error will occur. It may also be necessary to reason about how likely the error is to happen and the impact this has on the overall system behaviour. For example, for a randomised leader election protocol, the probability of message transmission failure can be modelled, or for a multimedia protocol the frame loss rate can be modelled [55]. A model of the system can be defined using a probabilistic model checking tool, and it is then possible to measure different aspects of performance, such as throughput, average response time and mean waiting time, while varying some system parameters. This is not possible using traditional model checking tools.

1.3 Parameterised Verification of Distributed Systems

Model checking provides the means to explore the *full* state space of a system model *automatically* and is therefore extremely useful in the verification of finite-state distributed systems. In spite of these benefits, model checking has its limitations. For example, if a system model (or property) has not been defined correctly it is possible to wrongly conclude that the behaviour of the system is correct. Note that this is an issue for any formal verification method. One of the constraints particular to model checking is that it is only possible to verify distributed systems with a fixed number of processes. So, if a property is verified for a given system with a set number of processes (identical up to renaming) and another process of the same type is added, it is necessary to carry out the verification again to prove that the property still holds. Ideally, it would be possible to verify a property for a system of a fixed size by model checking, and then prove that the property holds for that model with any given number of additional identical components. This is known as the *parameterised model checking problem* (PMCP).

The parameterised model checking problem is in general undecidable [4], but it is possible to verify parameterised systems, either by considering methods that are sound but incomplete or by restricting the set of systems that are analysed. An example of a common method for tackling the PMCP of the former type is the *network invariant* (a term first used in [62]) approach. In [62], the authors give an inductive proof rule (the *network invariant rule*) for a network invariant. If a finite-state invariant, I, can be constructed (manually) such that I satisfies the conditions of this rule, then any properties that are verified for the invariant will also hold for any particular size of the system under consideration. The conditions state that, for a single process P, $P \preceq I$ and $P||I \preceq I$, for some containment relation $\preceq$ and a parallel composition operator $||$. The work presented in [62] is based on a variant of CCS and CSP style specifications but can be used with any process theory given a set of conditions including monotonicity with respect to parallel composition. The authors give examples of a simple buffer and a token ring mutual exclusion protocol to demonstrate their approach. They also provide a general theorem stating that that there are instances of the PMCP for which finite-state invariants do not exist.

More recently, in [39] the authors outline a method for tackling the PMCP using network invariants. This approach is a partially automatic method for processes arranged in a ring. The authors provide a proof rule with conditions that allow the checking of the invariant rule to be discharged automatically, assuming that an abstraction mapping and an invariant are supplied. The conditions ensure that any fairness constraints are preserved under the abstraction mapping, thereby allowing liveness properties of the system to be considered. Examples of the approach are given for two dining philosophers protocols: a deterministic asymmetric system and a probabilistic symmetric one.

The network invariant approach is adopted in [19]. Here, the parameterised family of systems is described using a *network grammar* and the properties to be verified are specified in a *regular language*. An invariant is created using an abstraction method which groups states according to the atomic formula that they satisfy. It is shown that the invariant is greater under the simulation preorder (see Section 2.5) than any given system in the family and therefore any property that holds for the invariant will be satisfied for every member of the family of systems. By way of example, this method is applied to Dijkstra's token ring algorithm and to a binary tree algorithm for calculating the parity of the leaves, where each leaf has a binary value.

Another solution to the parameterised model checking problem is presented in [29] for arbitrary sized token rings. For certain forms of indexed CTL*\ X (see Section 2.3) properties it is shown that if the property holds for a ring of size less than some cutoff value then it will hold for any size of ring. Therefore model checking can be used to verify the property for a ring of a given size and it can then be deduced that the property holds for an arbitrary sized ring.

In general the network invariant approach only enables consideration of *safety* properties and is not strong enough to allow verification of *liveness* properties. Intuitively, this is due to the introduction of infinite behaviours, a result inherent to abstraction. In [53], a method is described based on *counter abstraction*. The approach is by counting the number of processes in a given *local* state, representing any count above 1 by the value 2. Each state in the concrete model is then represented in the abstract model by the counts of the process' local states. The authors' emphasis is on deriving fairness constraints that preserve liveness properties under the counter abstraction. Although the fairness requirements require some amount of manual derivation, heuristics are given which allow automation for certain types of property. The technique is applied to a number of examples including two algorithms for mutual exclusion. Note that this approach is limited to systems with a fully connected topology that use shared variables for communication.

In [52] an approach to the verification of safety *and* liveness properties of parameterised systems is described. The authors represent sets of reachable global states by regular expressions over a finite alphabet that represents the local state of a process. A transition relation between states is defined by a *transducer*, with properties to be verified also represented by regular expressions. Verifying that a property is an invariant of a system can be carried out in two ways: by forward exploration, starting from the initial state and iteratively considering the possible successor states; or by backward exploration, starting from each state that violates the property and iteratively considering the possible predecessor states. Neither of these are guaranteed to terminate for parameterised systems.

An accelerated transition, in which (an unbounded) number of processes perform a transition, can be used. This is a sound but incomplete method. Acceleration schemes must be deemed warranted and then selected manually but a number of "common" schemes are supplied: local acceleration (several transitions by one process); global acceleration of unary transitions (several processes make some transition, sequentially); global acceleration of binary transitions (several processes make some binary transition). Acceleration essentially combines a possibly unbounded number of applications of individual transitions into one "accelerated transition relation". Acceleration schemes are successfully applied to an example mutual exclusion system with synchronous communication. Note that the approach also deals with liveness properties by examining fairness constraints.

Much of the work on parameterised model checking assumes that the system has a regular topology. This is not the case in [12]. The authors define a simple concurrent language and describe how it can be used to describe parameterised programs. They employ an invariant approach but, rather than specifying an invariant for the system, they consider an invariant of a *property*. In particular they provide an invariance rule to prove that a state formula p is an invariant of some program P (i.e. always p is true over P) as follows:

1. The initial condition for P implies that some auxiliary assertion on states, z is true.
2. The assertion z implies p.
3. For every transition t of P, t and z implies z holds in the successor state for t.

The method of *invisible invariants* is employed such that z is derived automatically. By employing a colouring technique to deal with reachability predicates included in safety properties, it is possible to verify systems with irregular topologies.

In [22], a method for verifying parameterised systems is described that is based on *data independence*. A system is deemed data independent for a type if the type can be substituted by any other non-empty type and the operations on the type are restricted to input, storage, output and equality tests. For example a communication protocol that transmits data between processes is independent of the data content of the messages. Theorems for establishing a *threshold collection* for a data independent type are provided in [45]. If it can be shown that a property holds for every value in the threshold collection then the property is true for all values of the data independent type.

Data independence and induction are combined to provide a technique for the verification of systems parameterised by some data type, T, for simple topologies, in [22]. The method relies on the construction of an invariant that is also 'parameterised' by T. Induction is used to show that the invariant captures the behaviour of each system instantiated by a value in T. The data independence of T is then used to show that the property holds for the invariant for every value in some threshold collection of T. Discharging these two proofs implies that the property is true for every system instantiated by a value in T. The CSP process algebra is used to specify systems and the FDR model checker is used to reason about the systems. Two example protocols are verified using this technique. The method is extended in [23] to be applicable to more general network topologies, such as rings.

Finally, in [49] an approach used to verify the Firewire IEEE 1394 leader election protocol for any number of network nodes is described. The protocol enables the identification of a leader within a set of network nodes (connected in an acyclic topology). Each node can send *be my parent*, *be my child* requests or *acknowledgement* messages to neighbouring nodes. If a node receives *be my parent* requests from at least all but one of its neighbours then it responds to each of its neighbouring nodes with a *be my child* request. Once a node becomes a 'child' it sends an *acknowledgement* message to its 'parent' and then takes no further part in the protocol. This behaviour, whereby nodes 'drop-out' of the system, is termed *degenerative* [49].

The protocol is modelled and verified for fixed configurations using the *SPIN* model checker. The degenerative nature of the system is shown to be reflected in the underlying model of the system created from the SPIN specification. An inductive argument is then used and it is shown that the paths in the model of an arbitrarily sized system are *stutter-equivalent* (see Section 3.5.2) to the paths in the model of the 'degenerate' system. This implies that certain types of property will be satisfied in the model regardless of the number of network nodes considered.

1.4 Parameterised Verification of Randomised Distributed Systems

As for classical model checking, probabilistic model checking is restricted to verifying systems of a fixed size. Again, certain classes of system have been verified for an arbitrary number of processes: a survey of some of this work is provided in [50].

In [5] two methods for verifying liveness properties with probability 1 over parameterised probabilistic systems are presented. The first of these employs a *planner* to convert the probabilistic system to a non-deterministic one. Proving that a property holds in the nondeterministic system then guarantees that the same property will hold with probability 1 in the probabilistic one. Traditional solutions for non-probabilistic systems can then be used to verify that the property will hold for the nondeterministic version of the parameterised system. The second approach introduces a notion of γ-fairness. A (*simple*) temporal property will hold with probability 1 for a probabilistic system if and only if every γ-fair computation of the system satisfies that property. Again, methods for solving the non-probabilistic parameterised problem can then be employed.

In [27] the convergence of self-stabilising randomised protocols for a ring topology is considered. It is shown that, given a non-increasing measure on the state space of the model, if there exists a 'distance' measure between states and an ordering relation on the distance metric that satisfies certain conditions, then it is possible to deduce that the protocol will converge to some *legitimate* set of states with probability 1. They also provide a method to calculate the expected time of convergence by *lumping* the state space according to the distance measure.

The methods described above have only been applied to verification of *qualitative* properties i.e. properties that hold with probability 0 or 1. Parameterised model checking of *quantitative* properties has not been widely addressed, although some manual proofs of quantitative properties have been devised. For example, Aspnes and Herlihy [6] describe a weak shared coin protocol, that uses a shared counter which all processes can read from or write to. When a process reads the counter and it is above (below) a certain value the process will then choose to return 'heads' ('tails'). By appealing to results from random walk theory, they show that the probability of all processes returning heads (tails) is bounded below by the value $(K-1)/2K$, where $K \geq 1$ is an integer constant that is independent of the number of processes. Another manual correctness proof is given in the original paper describing the Itai Rodeh protocol [35] and in a further paper [30] describing simplifications of the Itai Rodeh protocol. Here it is shown that with probability 1 a unique leader will be elected for an arbitrary size of ring.

1.5 Thesis Outline and Contribution

In this thesis we explore the topic of parameterised model checking for state-based models of randomised distributed systems, adding to the body of previous work in this area, some of which is detailed in the previous sections. In Chapter 2 we begin with a review of the field of *model checking*, outlining background information relevant to the remainder of the thesis. In particular, we give a description of the use of state-based models and temporal logics in model checking. This is followed by a discussion of particular examples of model checking tools, and techniques that they exploit, before a review of some structural relations. In Chapter 3, *probabilistic model checking* is introduced. *Probabilistic* state-based models and temporal logics are discussed, before an overview of a probabilistic model checking tool is given. A survey of some probabilistic structural relations is provided at the end of this chapter. The work in these two chapters is standard from the literature.

In Chapter 4 we describe an approach to the PMCP using *invariants*, based on *data abstraction*, that can be used to tackle the parameterised model checking problem. The invariant method is applied to an example of a non-probabilistic system (a simple token ring protocol) in order to show how it can be used in practice. Thereafter a demonstration of how the technique can be applied to a probabilistic version of the token ring protocol is provided. The work contained in this chapter extends that of [17] through application of the technique to a new system and demonstrating that it can be applied to a probabilistic as well as a non-probabilistic form of the system. The probabilistic version of data abstraction described is similar to the abstraction via simulation of [24].

In Chapter 5 we describe an inductive proof schema for tackling the parameterised model checking of a particular class of systems, described as *deterministically degenerative*, using the Firewire IEEE 1394 Tree Identify Protocol to illustrate the technique. In Chapter 6 we extend this work, outlining a similar proof schema but for a wider class of probabilistic systems (described as probabilistically degenerative). We then describe the Itai Rodeh protocol and how it is modelled and verified using PRISM, explaining how the PMCP is tackled for the Itai Rodeh protocol using the proof schema. The work in these two chapters extends that of [49].

In particular, in Chapter 5 we generalise and extend the proof given for the IEEE 1394 Firewire protocol such that it is applicable to a class of probabilistic systems. The model specification for the Firewire IEEE 1394 protocol described in this chapter is based on the specification described in [49] but we have adapted it for the PRISM model checker and resolve root contention probabilistically rather than non-deterministically. The proof of correctness is again based on [49] but is adapted to take into account the probabilistic element of the models. The work in this chapter is also presented in [31].

The extension described in Section 5.7 has not been previously considered. In Chapter 6 we further extend the work described in Chapter 5. Although also based on the proof of correctness of the Firewire protocol given in [49], Theorem 6.1.7 is a new result. The PRISM specifications given for the Itai Rodeh leader election protocol described in this chapter are closely based on those given in [1] and outlined in [30]. The use of the inductive proof schema to prove correctness of this protocol is novel.

In Chapter 7 we introduce a class of protocols known as *gossip protocols* and describe a particular example and its analysis using probabilistic model checking. The parameterised model checking problem is then tackled for this protocol using an inductive approach. To our knowledge, the work described in this chapter is entirely new.

In Chapter 8 we describe some open problems and finally, Chapter 9 presents our conclusions.

1.6 Thesis Statement

We investigate the *parameterised model checking problem*, considering two approaches that have been successfully applied to *non-probabilistic* distributed systems. The first approach employs an *invariant* and is based on a form of *abstraction*; the second approach employs *induction* and is applicable to the class of *degenerative* systems. Both rely on establishing *structural relations* between *state-based models* of distributed protocols. We establish whether, using *probabilistic* structural relations between *probabilistic* state-based models, these approaches can be extended in order to be applicable to *randomised* distributed systems.

Summary In this chapter we have introduced the parameterised model checking problem and outlined some of the work that has been carried out on this to date for both probabilistic and non-probabilistic systems. We have outlined our contribution to this work, describing the contents of the thesis and providing a thesis statement.

Chapter 2

Modelling and Verifying Distributed Systems

Outline In this chapter we discuss some of the background to *model checking* that is relevant to the remainder of the thesis. We introduce *Kripke structures* and the *temporal logics*, *Computational Tree Logic* (CTL and CTL*) and *Linear Time Temporal Logic* (LTL). We also describe the verification techniques possible using the *SPIN* model checking tool, and outline the *Promela* modelling language. Finally, we consider the *bisimulation* and *simulation* structural relations for Kripke structures.

2.1 Introduction

The analysis of distributed systems is complex: unforeseen interactions between communicating processes can lead to errors in the design of such systems. Model checking provides the means to consider every combination of the executions of interacting processes (at an appropriate level of abstraction) and therefore is a useful verification technique when considering distributed systems. Model checking tools accept a (usually finite-state) model of a system specified by the user. Properties (usually defined using a temporal logic) are confirmed as being satisfied by the model, or not, in which case a counter-example to the property may be provided. A variety of model checking tools exist that employ different modelling and verification techniques. In this chapter we consider Kripke structures, the temporal logics CTL*, CTL and LTL and focus on the SPIN model checker. We consider these in particular since they are relevant to this thesis. For a more general introduction to model checking see, for example, [48].

Much of the work in the remainder of the thesis is concerned with verifying that a given property of a model of a system holds (by model checking) and then establishing a relation between the model and some other set of models. Structural relations between models are therefore integral to the thesis. We give a summary of some relevant relations over Kripke structures in the final part of this chapter.

2.2 Modelling Distributed Systems

A number of formalisms for modelling distributed systems exist. The majority of these are a variation of a state transition graph. One of the key features that all these graphs exhibit is that of *non-deterministic* choice. When modelling distributed systems we want to consider all possible interleavings of the executions of the processes: non-determinism enables this. We focus here on state-labelled transition graphs in the form of labelled Kripke structures.

Definition 2.2.1. *A (labelled) Kripke structure $\mathcal{K}$ is a tuple (S, S_0, R, L), where*

- *S is a finite set of states.*
- *$S_0 \subseteq S$ is the set of initial states.*
- *$R \subseteq S \times S$ is a total transition relation.*
- *$L : S \to 2^{AP}$ is a function that labels each state with a set of atomic propositions from AP, true in that state.*

The operation of a Kripke structure can be described in terms of *paths.* A finite path in a Kripke structure, $\mathcal{K} = (S, S_0, R, L)$, starting in state $s_0 \in S$, is a sequence of states $\pi = s_0, s_1, s_2, \ldots, s_n$, for $n \geq 0$, such that for all $0 \leq i \leq n$, $s_i \in S$ and for all $0 < i \leq n$, $(s_{i-1}, s_i) \in R$. An infinite path, $\pi = s_0, s_1, s_2, \ldots$ is defined similarly but for an infinite sequence of states. For an infinite path, $\pi = s_0, s_1, s_2, \ldots$, the path π^i is defined as the suffix of π, starting from state s_i. If a finite path ω is a prefix of a path π we write $\omega \leq \pi$

(or $\omega < \pi$ if it is a strict prefix). For a path, $\pi = s_0, s_1, s_2, \ldots, s_n$, the length of π, $|\pi|$ is $n + 1$. For an infinite path, $|\pi| = \infty$.

For any pair of states $s, t \in S$ such that $(s, t) \in R$ we say that (s, t) is a *transition* in $\mathcal{K}$, written $s \rightarrow_R t$ or $s \rightarrow t$ if R is clear from the context. Note that the transition relation for a Kripke structure is *total*, meaning that there must be at least one transition associated with each state. If, in the modelling of a system, a state s is defined with no outgoing transition we can ensure the transition relation is total by adding the *self-loop* transition, (s, s), to R. We describe any state s with no outgoing transitions, other than self-loops, as *terminal*.

The *trace* for a path π is given by the sequence of sets of atomic propositions that label each state in the path. If $\pi = s_0, s_1, \ldots, s_n$ is a finite path then $trace(\pi) = L(s_0), L(s_1), \ldots, L(s_n)$, and similarly if $\pi = s_0, s_1, \ldots$ is an infinite path then $trace(\pi) = L(s_0), L(s_1), \ldots$.

We generally consider Kripke structures with a single initial state. For a Kripke structure $\mathcal{K} = (S, S_0, R, L)$ with $S_0 = \{s_0\}$, we write $\mathcal{K} = (S, s_0, R, L)$.

We can consider a Kripke structure as a directed graph in which vertices represent states and edges transitions between states, as dictated by the transition relation. The labelling function associates each vertex with the set of atomic propositions that are true in that state. Indeed, we will often present Kripke structures diagrammatically in this form, with dashed lines indicating a transition between states, shown as circles on the diagram. Any labelling of a state will be included within the circle representing that state. Furthermore, we use the terms Kripke structure and *model* interchangeably throughout this thesis.

Example: Consider a single process participating in a mutual exclusion protocol that is *idle*, *trying* to enter its critical section, in its *critical* section or has *failed* to reach its critical section. We model this in two ways by the two Kripke structures given in Figure 2.1, using the single variable, *state*, to maintain the state of the process. Each of the states of $\mathcal{K}$ and $\mathcal{K}'$ are labelled by one of the propositions $state = idle$, $state = trying$, $state = critical$ or $state = failed$. The first Kripke structure assumes that the choice of whether a process enters its critical section or fails to do so is only made once the process tries to do so. In the second Kripke structure this outcome is pre-determined when the process moves from being idle to its trying state.

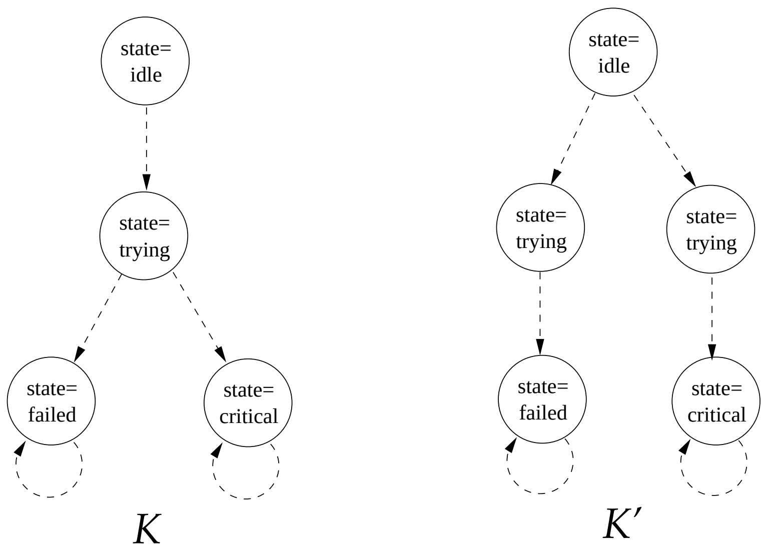


Figure 2.1: Kripke structures representing two versions of mutual exclusion for a single process

2.3 Temporal Logic

Temporal logics were first used by philosophers as a way of dealing with the progression of time in natural language arguments. They are an extension of modal logic and normally include some notion of 'always' in the future and 'eventually' in the future [18]. In terms of model checking, the semantics of temporal logic formulae are defined with respect to a Kripke structure. Therefore, a formula is described in terms of states, paths and atomic propositions. There are a number of different temporal logics used within different application domains. Three temporal logics that are commonly associated with model checking are defined below. These definitions are taken from [18].

2.3.1 Computational Treel Logic* (CTL*)

There are two path quantifiers in CTL*, defined as follows:

A The universal path quantifier. $\mathbf{A}\,\phi$ means for all paths ϕ holds.

E The existential path quantifier. $\mathbf{E}\,\phi$ means there exists some path along which ϕ holds.

The main temporal operators in CTL* are as follows:

G: The ‘always’ operator. $\mathbf{G}\,p$ means “p is *always* true in the present and at any point in the future”.

F: The ‘finally’ operator. $\mathbf{F}\,p$ means “p will eventually be true at some point in the future”.

X: The ‘next’ operator. $\mathbf{X}\,p$ means “p holds in the next state”.

U: The strong ‘until’ operator. $p\,\mathbf{U}\,q$ means “p holds until q holds and q must eventually hold”. Note that there is also a weak version of ‘until’, namely **W**, which is satisfied even if q does not eventually hold.

Formulae in CTL* are either state formulae or path formulae. CTL* is the set of all state formulae, where the syntax is defined by the following rules.

- If $p \in AP$ then p is a state formula.
- If f and g are state formulae, then $\neg f$, $f \wedge g$ and $f \vee g$ are state formulae.
- If f is a path formula then $\mathbf{E}\,f$ and $\mathbf{A}\,f$ are state formulae.
- If f is a state formula, then f is also a path formula.
- If f and g are path formulae, then $\neg f$, $f \vee g$, $f \wedge g$, $\mathbf{X}\,f$, $\mathbf{F}\,f$, $\mathbf{G}\,f$ and $f\,\mathbf{U}\,g$ are path formulae.

CTL* consists of all the state formulae defined by the above rules.

The semantics of CTL* are defined in terms of a Kripke structure, $\mathcal{K} = (S, S_0, R, L)$. If f is a state formula then for a state $s \in S$, $s \models f$ (s satisfies f) means that f is true at state s. If g is a path formula then $\pi \models g$ (π satisfies g) means that g is true along the path π of $\mathcal{K}$. Let $p \in AP$ and let f_1, f_2 be state formulae and g_1, g_2 path formulae, then the semantics of CTL* are defined inductively as follows.

1. $s \models p \Leftrightarrow p \in L(s)$
2. $s \models \neg f_1 \Leftrightarrow s \not\models f_1$
3. $s \models f_1 \vee f_2 \Leftrightarrow s \models f_1$ or $s \models f_2$

4. $s \models f_1 \wedge f_2 \Leftrightarrow s \models f_1$ and $s \models f_2$

5. $s \models \mathbf{E}\, g_1 \Leftrightarrow$ there is a path π from s such that $\pi \models g_1$

6. $s \models \mathbf{A}\, g_1 \Leftrightarrow$ for every path π starting from s, $\pi \models g_1$

7. $\pi \models f_1 \Leftrightarrow s$ is the first state of π and $s \models f_1$

8. $\pi \models \neg g_1 \Leftrightarrow \pi \not\models g_1$

9. $\pi \models g_1 \vee g_2 \Leftrightarrow \pi \models g_1$ or $\pi \models g_2$

10. $\pi \models g_1 \wedge g_2 \Leftrightarrow \pi \models g_1$ and $\pi \models g_2$

11. $\pi \models \mathbf{X}\, g_1 \Leftrightarrow \pi^1 \models g_1$

12. $\pi \models \mathbf{F}\, g_1 \Leftrightarrow$ there exists $k \geq 0$ such that $\pi^k \models g_1$

13. $\pi \models \mathbf{G}\, g_1 \Leftrightarrow$ for all $i \geq 0, \pi^i \models g_1$.

14. $\pi \models g_1\, \mathbf{U}\, g_2 \Leftrightarrow$ there exists $k \geq 0$ such that $\pi^k \models g_2$ and for all $0 \leq j < k, \pi^j \models g_1$

2.3.2 Computational Tree Logic (CTL)

CTL is a subset of CTL*, in which the operators are restricted to the form $\boldsymbol{QL}$, where $\boldsymbol{Q}$ is a CTL* path quantifier (**A** or **E**) and $\boldsymbol{L}$ a CTL* temporal operator (**F**, **G**, **X** or **U**). More formally, the syntax of CTL can be defined by restricting the rules for *path* formulae for CTL* to the following rule.

- If f and g are *state* formulae, then $\mathbf{X}\, f$, $\mathbf{F}\, f$, $\mathbf{G}\, f$ and $f\, \mathbf{U}\, g$ are path formulae.

2.3.3 Linear Time temporal Logic (LTL)

LTL (Linear Temporal Logic), like CTL, is a subset of CTL*. LTL is restricted to formulae of the form $\mathbf{A}f$, where f does not contain any path quantifiers. In general the operator **A** in state formulae is implicit and does not need to be included. In addition, **G** can be written as $\Box$ and **F** as $\Diamond$. The syntax of LTL path formulae can then be described by the following rules:

- If $p \in AP$, then p is a path formula.
- If f and g are path formulae, then $\neg f$, $f \vee g$, $f \wedge g$, $\mathbf{X}\, f$, $\Diamond f$, $\Box f$ and $f\, \mathbf{U}\, g$ are path formulae.

It is often useful to consider the subset of LTL that does not include the next-time operator which we denote by $\text{LTL}_{\setminus \mathcal{X}}$. Since we generally reason about events that occur at some point in the future rather than reasoning about the next step of a distributed algorithm, excluding this operator is not a great hardship. By considering this subset of LTL we have more flexibility in the analysis we can do.

CTL vs LTL The logics CTL and LTL are most commonly employed in model checking. Although CTL and LTL are both subsets of CTL*, they are not subsets of each other. Nonetheless, the majority of properties can be specified in both logics. When they cannot, sometimes there are ways around the problem. For example, a property of the form $\mathbf{E}\phi$, for a LTL path formula ϕ, cannot be expressed in LTL because LTL only allows formulae to be quantified over *all* possible execution paths whereas in this case it is necessary to show the *existence* of a path. However, observe that $\mathbf{E}\phi$ is equivalent to $\neg\mathbf{A}\neg\phi$. By finding a counterexample for $\mathbf{A}\phi$ in a model, the property $\mathbf{E}\phi$ can be established.

Linear Time vs. Branching Time Logic The temporal logics, CTL and CTL*, are described as *branching time* logics whilst LTL is described as a *linear time* logic. Branching time logic is quantified over paths from a state, whereas linear time logic is quantified over single computation paths.

Example: A classic illustration of the difference between branching and linear time logics is given in Figure 2.1. The two Kripke structures, $\mathcal{K}$ and $\mathcal{K}'$, show how a single process might act when trying to enter its critical section. The traces for the computation paths for both $\mathcal{K}$ and $\mathcal{K}'$ are given by,

$$state = idle, state = trying, state = critical, state = critical, \ldots \text{ and}$$
$$state = idle, state = trying, state = failed, state = failed, \ldots.$$

Both structures have the same set of traces, and therefore a linear time logic cannot distinguish between the two state graphs. However, in a branching time logic, it is possible to describe a property such as *if a process is trying to enter its critical section then it can.* More formally this would be written in CTL as

$$\mathbf{AG}(\mathit{state} = \mathit{trying} \rightarrow \mathbf{EX}\mathit{state} = \mathit{critical}).$$

This property would be satisfied by $\mathcal{K}'$ but not $\mathcal{K}$, so it is possible to differentiate between the two structures.

Safety vs Liveness Temporal logic properties can be classified by two types: *safety* properties and *liveness* properties. Safety properties must never be violated, while liveness properties are properties that a system must, at some point, satisfy [32]. So, for example a LTL property $\Diamond p$, where p is an atomic proposition, is a liveness property whereas $\Box p$ is a safety property.

2.4 Model Checking Tools & Techniques

One of the biggest problems that any model checking tool must combat is the *state space explosion* problem. This occurs because it is necessary to verify properties across *all* possible execution traces of a system model. Therefore, all possible interleavings of processes execution statements must be considered. This means that the state space of a model expands rapidly with the size of the system being modelled resulting in increased demands for resources. Eventually the demand for resources cannot be satisfied by the system running the model checker and complete verification is then no longer possible.

Different model checking tools use different techniques to address this problem. In this section we consider two model checkers and the methods they employ to tackle state space explosion. In particular, we describe the SPIN and SMV model checking tools. We make use of SPIN in Chapter 4 so we give more detail in this case, including only a brief outline of SMV for the sake of comparison.

2.4.1 SPIN

SPIN is a model checking tool that supports the *Pro*cess *me*ta *la*nguage, Promela. Promela is used to specify a model of a system, and SPIN can be used to verify LTL properties

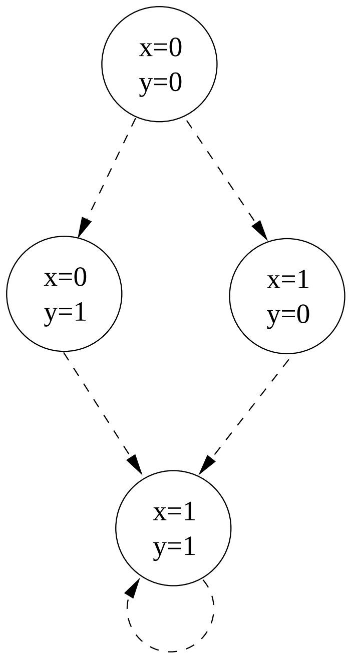


Figure 2.2: An example where partial order reduction can be exploited

of a Promela specification. We briefly outline some of the features of SPIN, examining the techniques it uses to reduce the time and memory requirements of verification before describing Promela.

The default algorithm that SPIN employs to traverse all the reachable states in the state graph during verification is essentially a depth-first search. However, the algorithm has been modified to allow properties to be verified and to permit on-the-fly verification. Verification *on-the-fly* means that the correctness of a property is checked at the same time as the state graph is built. Using this technique, it may not be necessary to construct the entire state space for two reasons. First, a counter-example for a property may be found before the entire state space has been constructed. Second, the property being checked 'guides' the construction of the state graph, and so any states that are not relevant to the property can be ignored. On-the-fly verification can therefore be useful in saving time and memory resources.

To combat the state space explosion problem, SPIN also avoids constructing the entire state space of a model, by exploiting a technique known as *partial order reduction.* A classic example of this is given in Figure 2.2. The Kripke structure shown in the figure represents an assignment to the variables `x` and `y`. The two paths of the model represent these assignments made in different orders (`x:=1;y:=1;` compared with `y:=1;x:=1;`). The final state is the same, regardless of the path taken, so the graph can be reduced by removing one of the paths. Note that a property is not preserved under partial order reduction if it contains the next time operator (**X**).

During verification, SPIN also allows the user to employ the *weak fairness* option. Weak fairness is used to ensure that no process in a specification is starved of execution time for an infinite time, i.e. that any process that can execute a statement will eventually execute it. This is useful because often, when performing a verification, a property will not hold because a particular process is never allowed to execute. Weak fairness, however, introduces a significant overhead to verification, particularly for systems with a large number of processes [32].

Promela

Before describing Promela it is important to note that modelling languages, such as Promela, are used to *specify* a system rather than to explicitly implement one. It is also important to distinguish between the *specification* of a system and the *model* of a system. A specification is a syntactically convenient way to represent the model. The model can therefore be derived from the specification. In the case of Promela specifications, the underlying model is given by a (representation of a) Kripke structure.

Promela has a C-like syntax, with only one type of module, defined by `proctype`. A `proctype` specification is given, and instantiations can then be executed either automatically, by declaring the process to be `active`, or manually by declaring an `init` process with a `run` statement for each process. Any number of processes can be executed concurrently, with a unique id being assigned to each process. As with a procedure, a process can take a set of parameters, although values can only be assigned to these when using `run`.

Data types are restricted and variables can only be defined as `mtype`, `unsigned`, `bit`, `int`, `short`, `byte`, `bool`, `chan` or `pid`. It is also possible to declare arrays of any of these types.

Most of the types are similar to the data types defined in C, but `mtype`, `chan` and `pid` require further explanation. The type, `pid`, is the simplest and is associated with the unique id assigned to each process. Note that it is possible to assign a variable to a process' id at runtime (e.g. `id1=run process1`).

The type `mtype` is an enumerated type, which can be assigned to include any set of labels. For example, it is possible to write `mtype={idle, trying, critical}`. A declaration `mtype state=idle`, then sets the variable `state` to be of `mtype`, initialised to `trying`.

An important feature of Promela is message passing along channels. A channel is declared to be of type `chan`. The length of the channel must also be declared. Declaring a channel with length greater than zero results in asynchronous message passing, but if a channel of length zero is defined then a receive and send on that channel will block until both are ready (*synchronised*). A type for each element of the channel must be declared too. Note that values of any type can be passed along channels, even channels themselves. Messages can be sent along channels using the `!` command or read from channels using the `?` command.

For example, the declaration `chan msg=[2] of {mtype, bool}` sets `msg` to be a channel of length two, with each message on that channel consisting of an enumerated type and a boolean. To pass a message of the form `msg!trying,true` along channel `msg`, the statement `msg!trying,true` is used. The statement `msg?state,end` can then be used to retrieve a message, where `end` is of type `bool`.

Promela also provides a selection and a repetition structure, respectively `if` and `do`. The code segments below give an example of each of these:

```
if
:: (done) -> out!msg;
:: (!done) -> msg--;
fi;

do
:: (in?1) -> out!3;
:: (in?2) -> out!4;
od;
```

Two important points about `if` and `do` statements in Promela are that selection is *non-deterministic* and the statement will *block* until at least one of the conditions is true. Non-deterministic choice means that if two conditions are true, then one will be selected arbitrarily.

Having blocking conditionals is also important as it provides a means whereby a process can wait for some condition to hold. Note that this is not unique to `if` or `do` statements. A conditional can in fact be inserted at any point in a process. For example the line `(state==trying);` would block, assuming `state` was not equal to `trying`, until some other process set `state` equal to `trying`.

One final useful feature that Promela provides is the `atomic` statement. This allows a section of code to be marked such that it will be executed as one statement, without being interrupted by other processes. For example `atomic{x=1;y=1;}`. Once this piece of code is executed, `x` and `y` will be set to one before any other process can execute. Note, however, that if any statement within an `atomic` statement blocks (for example a channel read where the channel is empty), then atomicity is broken and any other process may then execute.

2.4.2 SMV

SMV is a model checker with its own language component, that allows one to verify CTL properties using *Symbolic Verification*. SMV was designed mainly for verification of synchronous hardware circuits but can also be used to model check asynchronous concurrent software systems.

Symbolic Model Checking is implemented using Reduced Ordered Binary Decision Diagrams (ROBDD). A Binary Decision Diagram (BDD) is a tree-like representation of a boolean formula. The BDD, in Figure 2.3(a), represents the boolean formula $(P \wedge Q) \vee R$. Each path through this tree is an assignment of truth values to the variables P, Q and R, with the leaf nodes representing the value of the formula, given this assignment [18].

Binary Decision Diagrams are not a particularly concise representation of boolean formulae and contain a significant amount of redundancy. A *Reduced Ordered* Binary Decision Diagram (ROBDD) is a BDD in which the variables have been given some *ordering* and the BDD has been reduced to canonical form (up to ordering of the variables). An example of the ROBBD obtained from the Binary Decision Tree in Figure 2.3(a) is given in Figure 2.3(b) where the ordering of variables is $P < Q < R$. Note that the order of the variables is important in determining the size of the OBDD and that, although finding an optimal ordering is infeasible, there are heuristics available to find a reasonable one [18].

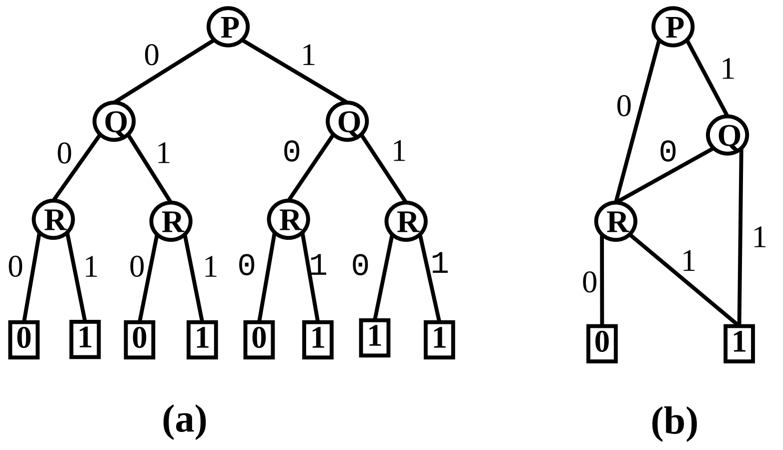


Figure 2.3: BDD and corresponding ROBDD for the formula $(P \wedge Q) \vee R$

Let $\mathcal{K} = (S, S_0, R, L)$ be a Kripke structure, then $\mathcal{K}$ can be represented by an ROBDD, if S, R and L are encoded using boolean vectors. The translation of a Kripke structure into an ROBDD can be done by first producing an explicit representation of $\mathcal{K}$ and then doing the encoding, but this is not necessarily practical due to the size of the explicit structure. Therefore, usually the ROBBD is created directly from a high level specification of the system [18].

The advantage of using the symbolic model checking technique described above is that ROBDDs can be an efficient and compact representation of a Kripke structure (subject to the variable ordering, as explained earlier). Therefore, this is a useful method for combating the state space explosion problem [18].

2.5 Model Relations

Sometimes, when analysing models, it is desirable to establish some form of *equivalence* between two models. In terms of model checking, equivalence between models generally means that they satisfy the same properties. For example, a model of a system may be too large to verify and one may therefore consider an appropriate *abstraction* that reduces the state space of the model. In order to ensure that properties that are true of the abstracted model will remain true in the original model we must establish some relation between the models. In this section we consider the bisimulation and simulation relations and discuss the logical characterisation of these with respect to LTL.

2.5.1 Bisimulation and Simulation

Bisimulation for Kripke structures is well-established (see for example [18]). Roughly speaking, two models are bisimilar if the stepwise behaviour of each model is matched in the other. Bisimulation preserves properties expressed in temporal logics (e.g. LTL). Note that there are two types of bisimulation, *strong* and *weak*. The weak form of the relation essentially ignores any transitions between related states (e.g. self-loops), whereas strong bisimulation does not. We only consider the strong form here.

Definition 2.5.1. *Given a Kripke structure $\mathcal{K} = (S, s_0, R, L)$ with atomic propositions, AP, an equivalence relation $H \subseteq S \times S$ is a bisimulation relation if and only if for all $s, s' \in S$ if $H(s, s')$ then*

1. *$L(s) = L(s')$*
2. *For every state $s_1 \in S$ such that $R(s, s_1)$, there is a state $s'_1 \in S$ with the property that $R(s', s'_1)$ and $H(s_1, s'_1)$.*
3. *For every state $s'_1 \in S$ such that $R(s', s'_1)$, there is a state $s_1 \in S$ with the property that $R(s, s_1)$ and $H(s_1, s'_1)$.*

Given two states p and q in S, p is bisimilar to q, written $p \approx q$, if and only if there is a bisimulation, H, such that $H(p, q)$. The bisimilarity relation, $\approx$, is an equivalence relation. Furthermore, it is the largest bisimulation relation over a given Kripke structure.

Lemma 2.5.2. *(See for example [18]) Let $\mathcal{K}$ be a Kripke structure with atomic propositions AP. Then for every LTL formula ϕ with atomic propositions in AP, $s \approx s'$ implies $s \models \phi \iff s' \models \phi$.*

In some instances bisimulation is too strict a relation. In other words it may be desirable to establish a relation between models that does not have as stringent conditions as bisimulation. In this instance a *simulation* relation may be appropriate. Whereas bisimulation is an equivalence relation, simulation is a preorder, hence it only guarantees preservation of properties between models in one direction. A *simulation relation* is defined formally as follows [18].

Definition 2.5.3. *Given two Kripke structures $\mathcal{K} = (S, s_0, R, L)$ and $\mathcal{K}' = (S', s'_0, R', L')$ with atomic propositions, AP and AP', respectively, such that $AP' \subseteq AP$, a relation $H \subseteq S \times S'$ is a simulation relation between $\mathcal{K}$ and $\mathcal{K}'$ if and only if $H(s_0, s'_0)$ and, for all $s \in S$ and $s' \in S'$, if $H(s, s')$ then*

1. *$L(s) \cap AP' = L'(s')$*
2. *For every state $s_1 \in S$ such that $R(s, s_1)$, there is a state $s'_1 \in S'$ with the property that $R'(s', s'_1)$ and $H(s_1, s'_1)$.*

$\mathcal{K}'$ is said to *simulate* $\mathcal{K}$, denoted $\mathcal{K} \preceq \mathcal{K}'$ if and only if there exists a simulation relation between $\mathcal{K}$ and $\mathcal{K}'$.

Intuitively, for a relation between two Kripke structures, $\mathcal{K}$ and $\mathcal{K}'$, to be a simulation the initial states must be in the relation and it must satisfy two conditions. Firstly for any two states that are related, the labelling of the state in $\mathcal{K}$ (restricted to the set of atomic propositions in $\mathcal{K}'$), must be equivalent to the labelling of the state in $\mathcal{K}'$. If this is true, secondly it is necessary to show, for any transition $s \to t$ in $\mathcal{K}$, that there exists a transition, $s' \to t'$ in $\mathcal{K}'$ such that the states s and s' are related and t and t' are related. If the transitions satisfy these conditions we say that $s \to t$ is *matched* by $s' \to t'$. If we can show this is true then the relation is a simulation relation.

Example: A simulation between two structures is illustrated in Figure 2.1, which shows two possible models for a single process participating in a mutual exclusion protocol. In this example $\mathcal{K}'$ simulates $\mathcal{K}$. This can be seen by choosing a simulation relation that associates each state in $\mathcal{K}'$ with each equivalently labelled state in $\mathcal{K}$. Note, however, that $\mathcal{K}$ does not simulate $\mathcal{K}'$. This can be seen by observing that from the state labelled by $state = trying$ in $\mathcal{K}$, it is possible to move to the states labelled $state = failed\ and$ $state = critical$. However, from one of the $state = trying$ labelled states in $\mathcal{K}'$, it is only possible to move to one of the states labelled $state = failed\ or\ state = critical$ but not both.

Lemma 2.5.4 is taken from [49], as adapted from [18].

Lemma 2.5.4. *Let $\mathcal{K}$ and $\mathcal{K}'$ be two Kripke structures with atomic propositions AP and AP' such that $AP' \subseteq AP$. Suppose that $\mathcal{K} \preceq \mathcal{K}'$. Then for every LTL formula ϕ with* *atomic propositions in AP', $\mathcal{K}' \models \phi$ implies $\mathcal{K} \models \phi$.*

Summary In this chapter we have considered the model checking problem, describing Kripke structures as a model for distributed systems and presenting the temporal logics CTL*, CTL and LTL. We have concentrated on SPIN as an example of a model checking tool, outlining some of the reduction techniques it employs in order to combat the state space explosion problem, and describing Promela. We have also defined the bisimulation and simulation relations for Kripke structures.

Chapter 3

Modelling and Verifying Randomised Distributed Systems

Outline In this chapter we describe the theory of probabilistic model checking. In particular, some elements of probability are discussed before an explanation of the theory of Markov chains. This is followed by a discussion of the logics Probabilistic CTL and Quantitative LTL. An overview of the probabilistic model checker, PRISM, is then given, including a description of the modelling language it uses. Finally, we examine some structural relations over probabilistic models.

3.1 Introduction

In the previous chapter we considered model checking, a verification technique that provides the means to formally establish properties of distributed systems. However, the focus of this thesis is on the analysis of *randomised* distributed systems and therefore we now consider *probabilistic model checking.* Similarly to traditional model checking tools, probabilistic model checkers accept a model specification and a property and verify that the property is true of the model. However, probabilistic model checking tools also allow probabilistic behaviour to be modelled and analysed. In this chapter we consider some aspects of probabilistic model checking, examining in particular those features relevant to the thesis. As discussed in Section 2.1, structural relations are fundamental to the results presented in subsequent chapters. In the previous chapter we defined relations on non-probabilistic models. We extend some of these and introduce some new relations for probabilistic models.

3.2 Probabilistic Models

Probabilistic model checking differs from traditional model checking in the representation of the model of a system. There are several types of probabilistic models but the ones that will be discussed here are all based on a *Markov chain.* There are two variants of the Markov chain model that we consider: Discrete Time Markov Chains and Markov Decision Processes. These are discussed in detail below. Other probabilistic models that are not considered here include Continuous Time Markov Chains and Probabilistic Timed Automata.

3.2.1 Discrete Time Markov Chains

A Discrete Time Markov chain (DTMC) can be viewed as a state transition system. A DTMC satisfies the *Markovian* property i.e. the choice of a transition to a new state is only determined by the current state. Transitions between states correspond to discrete time steps and have an associated probability. Note that the *state space* of a DTMC model is assumed to be discrete, and in the definition below to be *finite.* If we relax the first condition of Definition 3.2.1 to allow S to be countably infinite then we say that $\mathcal{D}$ is an *infinite state* DTMC. In order to analyse DTMCs, the states are labelled with a set of *atomic propositions* [55].

Definition 3.2.1. *(see for example [54]) A (labelled) Discrete Time Markov Chain is a tuple $\mathcal{D} = (S, S_0, \mathbf{P}, L)$, where*

- *S is a finite set of states,*
- *$S_0 \subseteq S$ is the set of initial states,*
- *$\mathbf{P} : S \times S \to [0,1]$ is a transition probability matrix such that, for all states $s \in S$, $\sum_{s' \in S} \mathbf{P}(s, s') = 1$,*

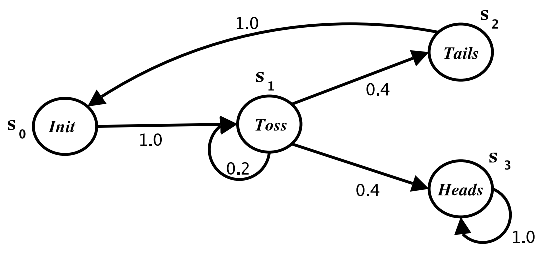


Figure 3.1: An example of a Discrete Time Markov Chain

- $L : S \rightarrow 2^{AP}$ *is a labelling function for a set of atomic propositions, AP.*

The matrix, $\mathbf{P}$, describes the probability of making a transition between any two states. For $s, s' \in S$, this probability is given by $\mathbf{P}(s, s')$. From the definition of $\mathbf{P}$, above, for every state the probabilities associated with the outgoing transitions from that state must sum to one. If some state s does not have an outgoing transition, we can ensure this condition is maintained by setting $\mathbf{P}(s, s) = 1$. If $\mathbf{P}(s, s') = p > 0$ we say that there is a transition from s to s' which we write as $s \xrightarrow{p} s'$ or just $s \rightarrow s'$.

An infinite *path*, ω, in a DTMC, $\mathcal{D}$, is a non-empty sequence s_0,s_1,... where for $i \geq 0$, $s_i \in S$. Similarly, a finite path, ω_{fin}, is a non-empty sequence s_0,s_1,...,s_n for some $n \geq 0$. The set of all infinite paths starting at state s is given by $\mathit{Path}(s)$ and the set of all finite paths starting at s by $\mathit{Path}_{fin}(s)$ [55].

Let $\omega_{\mathit{fin}}(i) = \omega(i) = s_i$ denote the ith state of a path and let $\mathit{last}(\omega_{\mathit{fin}}) = s_n$. Let $|\omega_{fin}|$ denote the length (the number of states) of a path (with $|\omega| = \infty$). By an abuse of notation, let $\mathbf{P}(\omega_{\mathit{fin}}) = \mathbf{P}(s_0, s_1) \times \mathbf{P}(s_1, s_2) \times \ldots \times \mathbf{P}(s_{n-1}, s_n)$ (with $\mathbf{P}(\omega_{\mathit{fin}}) = 1$ if $\omega_{\mathit{fin}} = s_0$).

Let $\mathit{trace}^{AP'}(\omega)$ denote the sequence given by the labelling of the states in ω restricted to the set of propositions in $AP' \subseteq AP$. For two paths, ω_{fin} and ω' with ω_{fin} finite, if ω is a prefix of ω' we write $\omega \leq \omega'$ (and $\omega < \omega'$ if it is a strict prefix). For states $s, t \in S$ we say that t is reachable from s if and only if there exists a path $\omega = s \rightarrow s_1 \rightarrow \ldots \rightarrow t$ such that $\omega \in \mathit{Path}_{\mathit{fin}}(s)$.

Note that we will generally consider DTMCs with a single initial state, s_0. In this instance, instead of writing $\mathcal{D} = (S, S_0, \mathbf{P}, L)$, we write $\mathcal{D} = (S, s_0, \mathbf{P}, L)$.

Example: Consider a simple coin-tossing process. Initially the process can move to a state from which it will either wait a time step, with probability 0.2, or it will toss a coin. In the latter case it will throw either a head or a tail, with equal probability. If it throws a tail, the process will restart and move back to the initial state, but if the process throws a head, it will move to a terminal state and cannot toss the coin again.

The DTMC in Figure 3.1 models this simple process (taken from [55]). The DTMC consists of four states, with $S = \{s_0, s_1, s_2, s_3\}$ and initial state, s_0. Each state is labelled with atomic propositions from the set {init, toss, heads, tails}. The transition probability matrix is given by:

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.2 & 0.4 & 0.4 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

A Probability Measure for DTMCs

For a finite path, α, of a DTMC, starting in state s define the *path cylinder* $\mathcal{C}(\alpha)$ by,

$$\mathcal{C}(\alpha) = \{\omega \in \mathit{Path}(s) | \alpha \leq \omega\}.$$

Then we can define the probability measure, Prob_s, on the smallest σ-field that contains all the sets $\mathcal{C}(\alpha)$ for all α, such that, $\mathit{Prob}_s(\mathcal{C}(\alpha)) = \mathbf{P}(\alpha)$. For more detail see, for example, [37].

3.2.2 Markov Decision Processes

The systems we seek to verify are randomised distributed protocols: concurrent systems for which individual processes can make a probabilistic choice. When describing these systems, we would like to model the scheduling of process statements non-deterministically (if the probabilities are unknown or irrelevant, we may also wish to model a probabilistic statement of a process abstractly as a non-deterministic choice). Hence, to model randomised distributed systems we require a model that exhibits both non-deterministic and probabilistic choice (note that DTMCs only exhibit probabilistic choice). We therefore consider *Markov Decision Processes* (MDPs), which can be considered as a generalisation of DTMCs. In particular, we consider state-labelled MDPs, where the states are augmented with a set of atomic propositions true in that state.

In the sequel, for any set Y, we let $Dist(Y)$ denote the set of all discrete probability distributions over Y. That is, $Dist(Y)$ is the set of all functions $\mu : Y \to [0,1]$ such that $\sum_{y \in Y} \mu(y) = 1$. For a set $X \subseteq Y$ and a distribution $\mu \in Dist(Y)$, we let $\mu(X) = \sum_{x \in X} \mu(x)$. Furthermore, for a distribution μ over Y, let $support(\mu) = \{y \in Y | \mu(y) > 0\}$.

Definition 3.2.2. *(See, for example, [54]). A (labelled) Markov Decision Process is a tuple $\mathcal{M} = (S, S_0, Steps, Act, L)$ where,*

- *S is a finite set of states,*
- *$S_0 \subseteq S$ is the set of initial states,*
- *Act is a set of actions,*
- *$Steps : S \to 2^{Act \times Dist(S)}$ is the probabilistic transition function such that*

$$\forall s \in S, Steps(s) \neq \emptyset,$$

- *$L : S \to 2^{AP}$ is a labelling function over a set of atomic propositions AP.*

The main differences between a MDP and a DTMC is the addition of an action set, *Act* and the replacement of the two-dimensional probability transition matrix with *Steps*. For MDP, $\mathcal{M} = (S, S_0, Steps, Act, L)$, the function, *Steps*, maps each state in S to a non-empty subset of $Act \times Dist(S)$, where $Dist(S)$ is the set of all probability distributions over S. Intuitively, for some state $s \in S$, *Steps* makes a non-deterministic choice over $|Steps(s)|$ possible (action, distribution) pairs, choosing action a and distribution μ, say. According to the distribution, μ, a probabilistic choice is made over the states of the model where the probability of moving to a state s' is given by $\mu(s')$.

For a state s and an (action, distribution) pair $(a, \mu) \in Steps(s)$, we say that a is *enabled* from s. If $\mu(s') > 0$ for some state s' we say that there is a transition from s to s' which we write as $s \xrightarrow{a,\mu} s'$ or just $s \to s'$ if it is clear which element of $Steps(s)$ we are considering. A transition $s \xrightarrow{a,\mu} s'$ is *non-probabilistic* if $\mu(s') = 1$ (otherwise we describe a transition as *probabilistic*). A transition $s \xrightarrow{a,\mu} s'$ *stutters* if and only if $L(s) = L(s')$.

An action $a \in Act$ is *non-probabilistic* if and only if, for every $s \in S$, $\forall (a, \mu) \in Steps(s)$, $\mu(s') = 1$ for some $s' \in S$. We describe $\mathcal{M}$ as *non-deterministic* if every action in *Act* is non-probabilistic. Action $a \in \mathit{Act}$ is a *stutter* action if and only if, for all $s \in S$, for every $(a, \mu) \in \mathit{Steps}(s)$, $\mu(s') > 0 \implies L(s) = L(s')$.

An infinite *path*, ω in $\mathcal{M}$ is a non-empty sequence s_0,a_0,μ_0,s_1,a_1,μ_1,... where for $i \geq 0$, $s_i \in S$, $(a_i, \mu_i) \in \mathit{Steps}(s_i)$, $\mu(s_{i+1}) > 0$. We write this as,

$$s_0 \xrightarrow{a_0,\mu_0} s_1 \xrightarrow{a_1,\mu_1} \ldots$$

Similarly, a finite path, ω_{fin}, is a non-empty sequence, s_0,a_0,μ_0,s_1,a_1,μ_1,...,a_{n-1},μ_{n-1},s_n for some $n \geq 0$ which we write as,

$$s_0 \xrightarrow{a_0,\mu_0} s_1 \xrightarrow{a_1,\mu_1} \ldots \xrightarrow{a_{n-1},\mu_{n-1}} s_n.$$

The set of all infinite paths starting at state s is given by $\mathit{Path}(s)$ and the set of all finite paths starting at s by $\mathit{Path}_{fin}(s)$.

Let $\omega_{\mathit{fin}}(i) = \omega(i) = s_i$ denote the ith state of a path and let $\mathit{last}(\omega_{\mathit{fin}}) = s_n$. Let $|\omega_{fin}|$ denote the length (the number of actions) of a path (with $|\omega| = \infty$). By an abuse of notation, let $\mathbf{P}(\omega_{\mathit{fin}}) = \mathbf{P}(s_0, s_1) \times \mathbf{P}(s_1, s_2) \times \ldots \times \mathbf{P}(s_{n-1}, s_n)$ (with $\mathbf{P}(\omega_{\mathit{fin}}) = 1$ if $\omega_{\mathit{fin}} = s_0$).

Let $\mathit{trace}^{AP'}(\omega)$ denote the sequence given by the labelling of the states in ω restricted to the set of propositions in $AP' \subseteq AP$ and let $\mathit{act}^{\mathit{Act}'}(\omega)$ denote the action sequence for ω restricted to the actions in $\mathit{Act}' \subseteq \mathit{Act}$.

For two paths, ω_{fin} and ω' with ω_{fin} finite, if ω is a prefix of ω' we write $\omega \leq \omega'$ (and $\omega < \omega'$ if it is a strict prefix). For states $s, t \in S$ we say that t is reachable from s if and only if there exists a path $\omega = s \to s_1 \to \ldots \to t$ such that $\omega \in \mathit{Path}_{\mathit{fin}}(s)$.

Furthermore, for $(a, \mu) \in \mathit{Steps}(\mathit{last}(\omega))$, $t \in S$ we let $\omega \xrightarrow{a,\mu} t$ denote the finite path,

$$s_0 \xrightarrow{a_0,\mu_0} s_1 \xrightarrow{a_1,\mu_1} \ldots \xrightarrow{a_{n-1},\mu_{n-1}} s_n \xrightarrow{a,\mu} t.$$

Note that we generally consider MDPs with a single initial state, s_0. In this instance, instead of writing $\mathcal{M} = (S, S_0, \mathit{Steps}, \mathit{Act}, L)$, we write $\mathcal{M} = (S, s_0, \mathit{Steps}, \mathit{Act}, L)$. Similarly, we will sometimes consider an empty action set $\mathit{Act} = \emptyset$. In this instance, instead of writing $\mathcal{M} = (S, S_0, \mathit{Steps}, \mathit{Act}, L)$, we write $\mathcal{M} = (S, s_0, \mathit{Steps}, L)$ and let $\mathit{Steps} : S \to \mathit{Dist}(S)$.

Adversaries

In order to analyse a MDP we need to resolve the non-determinism. This is done by considering *adversaries*, constructs that make a choice over the set $Steps(s)$ for each state s of a MDP, based on the history of choices.

Definition 3.2.3. *[55] A* deterministic adversary *A of a MDP $\mathcal{M} = (S, s_0, Steps, Act, L)$ is a function mapping every finite path ω onto an element $A(\omega)$ of the set $Steps(last(\omega))$.*

In the sequel we let $Adv_{\mathcal{M}}$ denote the set of all possible adversaries of the MDP $\mathcal{M}$ and, for any adversary A and state s, we let $Path^A(s)$ denote the subset of $Path_s$ which corresponds to A, and similarly $Path^A_{fin}(s)$ is the subset of $Path_{fin}(s)$ that corresponds to A [55].

Each adversary of a MDP produces an infinite-state DTMC, with each state in the DTMC given by the history of states so far visited. The following definition is adapted from [55].

Definition 3.2.4. *For a given adversary, A of a MDP $\mathcal{M} = (S, s_0, Steps, Act, L)$ and a state $s \in S$, the behaviour from s under A is given by the infinite state DTMC induced by A, $\mathcal{D}^A = (S^A, s_0^A, \mathbf{P}^A, L^A)$ where,*

- $S^A = Path^A_{fin}(s)$
- $s_0^A = s$
- *For* $\omega, \omega' \in S^A$, $\mathbf{P}^A(\omega, \omega') = \begin{cases} \mu(s') & \text{if } \omega' \equiv \omega \overset{A(\omega)}{\rightarrow} s' \text{ and } A(\omega) = (a, \mu) \\ 0 & \text{otherwise} \end{cases}$
- *For all* $\omega \in S^A$, $L^A(\omega) = L(last(\omega))$.

Since an adversary uniquely determines a DTMC of the form described above, in the sequel it will be convenient to refer to an adversary of a MDP when referring to the infinite state DTMC induced by the adversary.

Example: Consider the MDP given in Figure 3.2 (taken from [55]). The figure shows a four state MDP (note that the labelling of the states is omitted). The dotted lines represent a nondeterministic transition from a state and the solid lines represent the probabilistic distribution reached from a non-deterministic transition. Note that the only state from which it is necessary to make a non-deterministic choice in this MDP is state 1. We denote

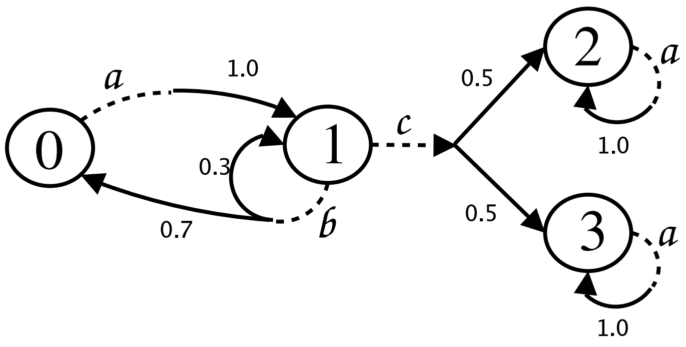


Figure 3.2: A Markov Decision Process

the two probability distributions reachable from state 1 as μ_b and μ_c (where $\mu_b(0) = 0.7$, $\mu_b(1) = 0.3$ and $\mu_c(2) = \mu_c(3) = 0.5$) and the trivial distributions reachable from states 0, 2 and 3 as μ_a, μ_d and μ_e respectively. We can define two example adversaries as follows [55]:

- Adversary A_1 : $A_1(0) = (a, \mu_a)$, $A_1(01) = (c, \mu_c)$, $A_1(012) = (d, \mu_d)$, $A_1(013) = (e, \mu_e)$, $\ldots$ etc.
- Adversary A_2 : $A_2(0) = (a, \mu_a)$, $A_2(01) = (b, \mu_b)$, $A_2(011) = (c, \mu_c)$, $A_2(010) = (a, \mu_a)$, $A_2(0101) = (c, \mu_c)$, $A_2(0112) = (d, \mu_d)$, $A_2(0113) = (e, \mu_e)$, $A_2(01012) = (d, \mu_d)$, $A_2(01013) = (e, \mu_e)$, $\ldots$, etc.

Fragments of the infinite state DTMCs, $\mathcal{D}^{A_1}$ and $\mathcal{D}^{A_2}$, that are derived from the MDP given in Figure 3.2 under the adversaries, A_1 and A_2, described above, are shown in Figure 3.3 [55].

Randomised Adversaries

In the above we considered history-dependent *deterministic* adversaries that, for a given finite path, select a single (action, distribution) pair from the set enabled from the last state of the path. We now consider adversaries that make a choice according to some *distribution* (see for example [57]).

Definition 3.2.5. *A* randomised adversary *E of a MDP $\mathcal{M} = (S, s_0, Steps, Act, L)$ is a function mapping every finite path ω onto an element $E(\omega)$ of $Dist(Steps(last(\omega)))$.*

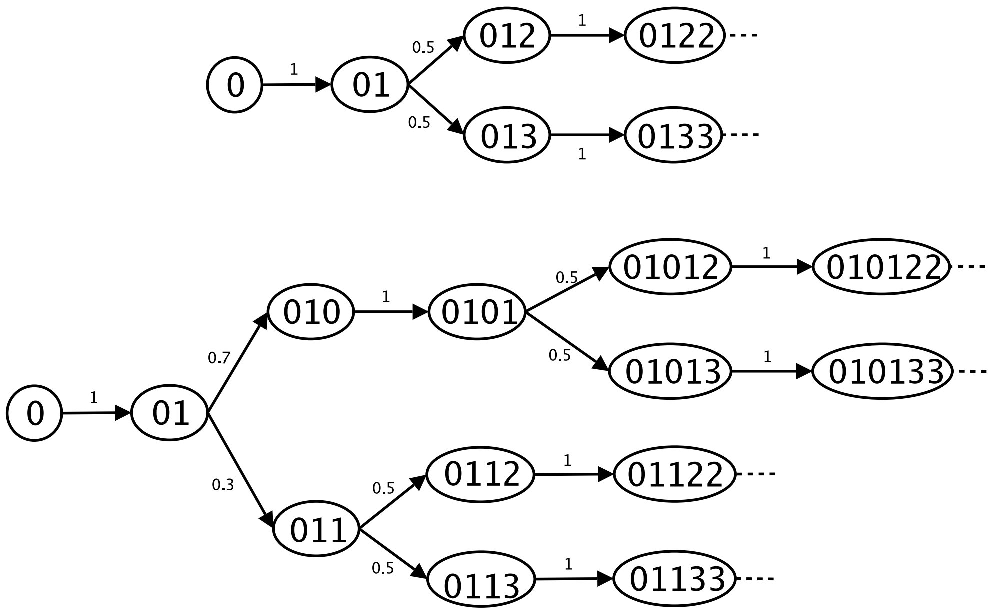


Figure 3.3: Portions of two infinite state DTMCs resulting from the behaviour of a MDP under adversaries, A_1 (top) and A_2 (bottom)

In the sequel we let $RAdv_{\mathcal{M}}$ denote the set of all possible randomised adversaries of the MDP $\mathcal{M}$ and, for any adversary E and state s, let $Path^E(s)$ denote the subset of $Path(s)$ which corresponds to E.

A Probability Measure for analysing MDPs

We can extend the definition of the probability measure over DTMCs from Section 3.2.1 to infinite state DTMCs in a straightforward manner. Furthermore, there exists a one-to-one mapping between the paths in the infinite state DTMC generated by an adversary of a MDP and the paths in a MDP. For a finite path, α, starting in state s of a MDP, $\mathcal{M} = (S, S_0, Steps, Act, L)$, the *path cylinder* $\mathcal{C}(\alpha)$, under an adversary A of $\mathcal{M}$, is defined by [55],

$$\mathcal{C}(\alpha) = \{\omega \in Path^A(s) | \alpha \leq \omega\}.$$

Then we can define the probability measure, $Prob_s^A$, on the smallest σ-field that contains all the sets $\mathcal{C}(\alpha)$ for all α, such that, $Prob_s^A(\mathcal{C}(\alpha)) = \mathbf{P}(\alpha)$. For more detail see, for example, [14, 37].

Similarly, for a randomised adversary, E, of $\mathcal{M}$, we can define the probability measure,

$Prob_s^E$, on the smallest σ-field that contains all the sets $\mathcal{C}(\alpha)$ for all finite paths α, such that,

$$Prob_s^E(s) = 1$$

and, for $\alpha = \beta \xrightarrow{a,\mu} t$ (for a finite path β, $(a, \mu) \in Steps(last(\beta))$ and $t \in S$),

$$Prob_s^E(\mathcal{C}(\alpha)) = Prob_s^E(\mathcal{C}(\beta)) \times E(\beta)((a, \mu)) \times \mu(t).$$

For more detail see, for example, [57].

Cuts

It is often useful to consider a finite portion of the Markov chain induced by an adversary. Intuitively, a *cut*, as defined below, is a set of finite paths of an adversary of a MDP. Each path in the cut starts from the initial state of the MDP, no path in the cut is the prefix of another path in the cut and every finite path under the adversary is the suffix or prefix of a path in the cut. Note that a cut is a simplification of a *fringe* as defined for probabilistic automata by Segala [56].

Definition 3.2.6. *Let $\mathcal{M}$ =(S,s_0,Steps,Act,L) be a MDP and let $A \in Adv_{\mathcal{M}}$. Define Cut(A) to be a family of sets s.t. for $D \in Cut(A)$, $D \subseteq Path_{fin}^A(s_0)$ where, for all $\omega \in D$, $\omega \nleq \omega'$ and $\omega' \nleq \omega$ for any $\omega' \in D$, $\omega' \neq \omega$ and,*

$$\sum_{\omega \in D} Prob_{s_0}^A(\mathcal{C}(\omega)) = 1.$$

Given an adversary A of a MDP, for $n \geq 0$, let $cut^A(n) \in Cut(A)$ be defined such that for all $\omega \in cut^A(n)$, $|\omega| = n$. For $D \in Cut(A)$ we say that D is a *cut* of A. Furthermore, we describe $cut^A(n)$ as a cut of A *at depth n*.

Example: In Figure 3.3 we presented portions of two infinite state DTMCs resulting from the behaviour of the MDP in Figure 3.2, under adversaries A_1 and A_2. In Figure 3.4 we repeat the infinite state DTMC obtained under A_2 and include two examples of cuts of A_2, D_1 and D_2. These are represented by dashed lines in the diagram. The cut D_1 is a cut of A_2 at depth three, and is given by the set $\{010, 011\}$ whilst D_2 is given by the set $\{01012, 01013, 0112, 0113\}$. It should be clear that D_1 and D_2 both satisfy the conditions of Definition 3.2.6.

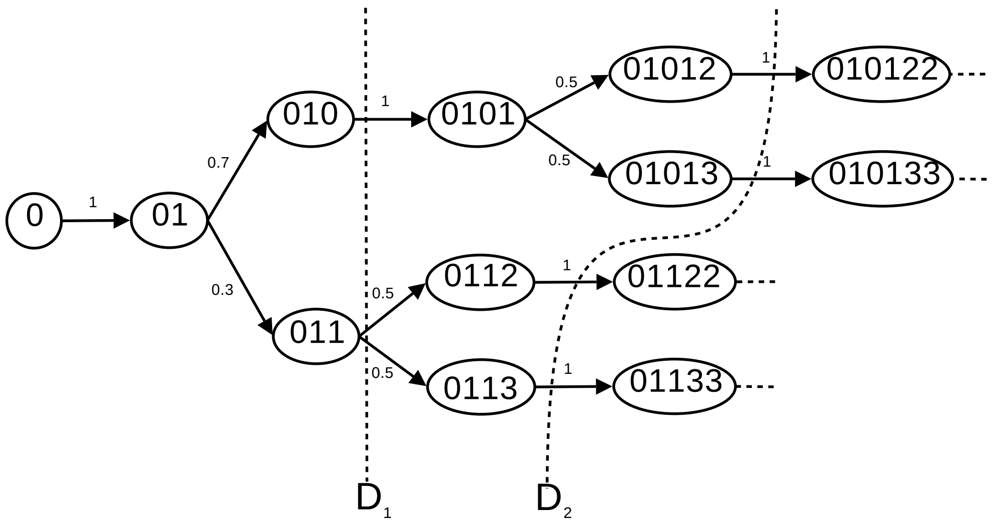


Figure 3.4: Examples of cuts, D_1 and D_2, of the adversary, A_2 (see Figure 3.3)

3.3 Probabilistic Temporal Logics

There are several possible logics that can be employed in probabilistic model checking. The two that are described here are based on the CTL and LTL temporal logics commonly used in traditional model checking (see Section 2.3).

3.3.1 Probabilistic Computation Tree Logic

PCTL (Probabilistic CTL) is a probabilistic extension of the temporal logic, CTL. It can be used to specify properties to be verified for a DTMC or a MDP. It should be noted, however, that the semantics of PCTL for a DTMC are slightly different from that for a MDP and hence these are dealt with separately below.

PCTL for DTMCs

PCTL formulae are defined over states within a model. The formal syntax for PCTL state formula (ϕ) are given by the following rules [55].

$$\phi ::= \text{ true} \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \mathcal{P}_{\bowtie p}[\psi]$$

where a is an atomic proposition, $\bowtie \in \{\leq, <, >, \geq\}, p \in [0,1]$ and ψ is a path formula given by,

$$\psi ::= \ \phi \mid \mathcal{X}\phi \mid \phi\mathcal{U}^{\leq k}\phi \mid \phi\mathcal{U}\phi$$

such that $k \in \mathbb{N}$.

The next-time ($\mathcal{X}$) and until ($\mathcal{U}$) operators are as for the CTL counterparts (**X** and **U** respectively). For the time-bounded until operator ($\mathcal{U}^{\leq k}$), a path satisfies $\phi_1\mathcal{U}^{\leq k}\phi_2$ if ϕ_1 is true along the path until ϕ_2 is true and ϕ_2 becomes true within k time steps. For a DTMC, some state, s, satisfies $\mathcal{P}_{\bowtie p}[\psi]$, if the probability over the set of paths starting in s that satisfy path formula ψ is within the bounds specified by $\bowtie p$.

The formal semantics of PCTL for any DTMC, $\mathcal{D} = (S, s_0, \mathbf{P}, L)$, are given in terms of paths and states. For any state $s \in S$ and for any infinite path, π, the formal semantics of PCTL can be defined inductively in terms of the satisfaction relation, $\models$, as follows [55].

1. $s \models \text{true}, \forall s \in S$
2. $s \models a \iff a \in L(s)$
3. $s \models \neg\phi \iff s \not\models \phi$
4. $s \models \phi_1 \wedge \phi_2 \iff s \models \phi_1 \wedge s \models \phi_2$
5. $s \models \mathcal{P}_{\bowtie p}[\psi] \iff \mathit{Prob}_s(\{\pi \in \mathit{Path}(s) | \pi \models \psi\}) \bowtie p$
6. $\pi \models \phi \iff \pi(1) \models \phi$
7. $\pi \models \phi_1\mathcal{U}^{\leq k}\phi_2 \iff \exists i \leq k. \, (\pi(i) \models \phi_2 \wedge \pi(j) \models \phi_1, \, \forall j < i)$
8. $\pi \models \phi_1\mathcal{U}\phi_2 \iff \exists k \geq 0 \, . \, \pi \models \phi_1\mathcal{U}^{\leq k}\phi_2$

The probability measure over paths in $\mathit{Path}(s)$, Prob_s, is as defined in Section 3.2.2. Note that the set of paths defined by a PCTL path formula is measurable [61]. For a PCTL property, ϕ, in the remainder of the thesis, we let $\mathcal{D} \models \phi$ if and only if $s_0 \models \phi$ where s_0 is the initial state of $\mathcal{D}$.

The eventually ($\Diamond$) and always ($\Box$) operators are not included in the syntax for PCTL but we can use the following equivalences to define them. The $\Diamond$ operator can be defined in terms of the until operator, since:

$$\mathcal{P}_{\bowtie p}[\Diamond\phi] \equiv \mathcal{P}_{\bowtie p}[\text{true}\,\mathcal{U}\,\phi].$$

Notice that $\Box\phi \equiv \neg\Diamond\neg\phi$. Since PCTL does not allow path formulae to be negated, this equivalence cannot be used directly, but

$$Prob_s(\{\omega \in Path(s) | \omega \models \neg\psi\}) = 1 - Prob_s(\{\omega \in Path(s) | \omega \models \psi\}),$$

and so:

$$\mathcal{P}_{\bowtie p}[\Box\,\phi] \equiv \mathcal{P}_{\overline{\bowtie}(1-p)}[\Diamond\,\neg\,\phi],$$

where $\overline{\leq} \equiv\geq, \overline{<} \equiv>, \overline{\geq} \equiv\leq, \overline{>} \equiv\leq$. Note that since the always and eventually operators are defined in terms of the until ($\mathcal{U}$) operator, there are equivalent time-bounded always and eventually operators that can be defined by replacing $\mathcal{U}$ by $\mathcal{U}^{\leq k}$ in the above [55].

In CTL, the always and eventually operators cannot be used individually in a formula. They must always be paired with a CTL* path quantifier, for all paths (**A**) or there exists a path (**E**). The equivalent restriction in PCTL is that path formulae can only be included as a parameter of the $\mathcal{P}_{\bowtie p}$ operator. Therefore, in a loose sense, the $\mathcal{P}_{\bowtie p}$ operator can be considered the analogue of the **A** and **E** CTL* operators. For example $\mathbf{EF} \equiv \mathcal{P}_{>0}[\Diamond\phi]$ but note that not all CTL operators have direct equivalences [55].

In the subsequent chapters, in order to establish results for parameterised probabilistic systems, it is necessary to restrict the set of properties that we verify. We consider two such restrictions: PCTL-liveness and PCTL-reachability.

Definition 3.3.1. *[9] A PCTL-liveness formula, ϕ is defined to be a PCTL formula that contains only the operator, $\mathcal{P}_{>p}$, and for which negation ($\neg$) is only applied to atomic propositions.*

Definition 3.3.2. *A PCTL-reachability formula, ϕ, is defined to be a PCTL formula of the form $\mathcal{P}_{\underline{\bowtie} p}[\Diamond\psi]$, for $\underline{\bowtie} \in \{\geq, \leq\}$ and ψ ::= true | a | $\psi \wedge \psi$ where a is an atomic proposition.*

Consider the set of atomic propositons, $\{x = 0, x = 1, y = 1, y = 2\}$. The property,

$$\mathcal{P}_{>0}[x = 0\,\mathcal{U}\,(\neg y = 1)],$$

is a PCTL-liveness property whereas,

$$\mathcal{P}_{\leq 0.5}[\Diamond(x = 1 \land y = 1)],$$

is a PCTL-reachability property.

PCTL for MDPs

As mentioned earlier, the definition of PCTL for DTMCs and MDPs is slightly different. The syntax is identical for both types of models and the semantics for the path operators is the same but the $\mathcal{P}_{\bowtie p}$ operator is defined differently. In order to analyse the probability of some property holding we must first resolve the non-determinism by means of the adversaries. Therefore, we must consider the probability over the set of all adversaries. Specifically, $\mathcal{P}_{\bowtie p}[\psi]$ holds for some state of a MDP if the probability of ψ being satisfied meets the bound $\bowtie p$ *for all resolutions of non-determinism.* Formally we can specify the syntax and semantics of PCTL for a MDP, $\mathcal{M}$, as per the definition in the previous section but replacing the definition of $\models$ for $\mathcal{P}_{\bowtie p}$ by:

$$s \models \mathcal{P}_{\bowtie p}[\psi] \iff Prob_s^A(\{\omega \in Path^A(s) | \omega \models \psi\}) \bowtie p, \, \forall A \in Adv_{\mathcal{M}}.$$

As for a DTMC, the set of paths under an adversary of a MDP, defined by a PCTL path formula, is measurable [61]. For a PCTL property ϕ, we let $\mathcal{M} \models \phi$ if and only if $s_0 \models \phi$ where s_0 is the initial state of $\mathcal{M}$.

3.3.2 Quantitative Linear Time Logic

In Section 2.3 we introduced the logic LTL for reasoning about non-probabilistic models. Recall that a LTL property is defined over paths of a (non-probabilistic) model and, in particular, we reason over *every* path. It is straightforward to adapt the definition of a LTL path formula to paths in a MDP. We now consider a probabilistic variant of LTL which we describe as *quantitative* Linear Time Logic (QLTL).

A QLTL (QLTL$_{\setminus\mathcal{X}}$) formula ϕ is defined over states of a MDP with syntax given by $\phi ::= \mathcal{P}_{\bowtie p}[\psi]$, where $\bowtie \in \{\leq, <, >, \geq\}, p \in [0, 1]$ and ψ is a LTL (LTL$_{\setminus\mathcal{X}}$) path formula. For a MDP, $\mathcal{M}$, a state s of $\mathcal{M}$, and a quantitative LTL property, $\phi = \mathcal{P}_{\bowtie p}[\psi]$, we say that s satisfies ϕ, denoted $s \models \phi$, if, for every adversary A of $\mathcal{M}$,

$$Prob_s^A(\{\omega \in Path_s^A | \omega \models \psi\}) \bowtie p.$$

We say that $\mathcal{M}$ satisfies ϕ, denoted $\mathcal{M} \models \phi$ if and only if $s_0 \models \phi$ where s_0 is the initial state of $\mathcal{M}$. The set of paths of a MDP under an adversary that satisfy a LTL path formula are measurable by $Prob_s^A$ [61].

3.3.3 Randomised Adversaries and Probabilistic Temporal Logics

Above we considered the definition of the semantics of QLTL and PCTL with respect to deterministic adversaries. Observe that the randomised adversaries of a MDP are, in a sense, probabilistic combinations of their deterministic counterparts. When reasoning about a PCTL or QLTL property it is sufficient to consider the maximum or minimum probability with which some property is satisfied. These probabilities can be observed by considering the deterministic adversaries of a MDP. Equivalently, they can be observed over the randomised adversaries since probabilistic combination cannot increase (decrease) the maximum (minimum) probability. Specifically, for a MDP $\mathcal{M}$ and a PCTL property ϕ, it has been shown that ϕ is satisfied by $\mathcal{M}$ under the set of deterministic adversaries of $\mathcal{M}$ if and only if $\mathcal{M}$ satisfies ϕ under the set of randomised adversaries of $\mathcal{M}$ [56, 58]. From this we can also conclude that,

Lemma 3.3.3. *For a MDP $\mathcal{M}$ and a QLTL property ϕ, ϕ is satisfied by $\mathcal{M}$ under $Adv_{\mathcal{M}}$ if and only if $\mathcal{M}$ satisfies ϕ under $RAdv_{\mathcal{M}}$.*

From [56, 58], we also have that,

Lemma 3.3.4. *Let $\mathcal{M} = (S, s_0, Steps, Act, L)$ be a MDP. Given a LTL path formula ψ, for every randomised adversary E of $\mathcal{M}$, there exist deterministic adversaries $\bar{A}$ and $\underline{A}$ of $\mathcal{M}$ such that,*

$$Prob_{s_0}^E(\{\omega \in Path_{s_0}^E | \omega \models \phi\}) \geq Prob_{s_0}^{\underline{A}}(\{\omega' \in Path_{s_0}^{\underline{A}} | \omega' \models \phi\}) \text{ and,}$$

$$Prob_{s_0}^E(\{\omega \in Path_{s_0}^E | \omega \models \phi\}) \leq Prob_{s_0}^{\bar{A}}(\{\omega' \in Path_{s_0}^{\bar{A}} | \omega' \models \phi\}).$$

3.4 The PRISM Probabilistic Model Checking Tool

The Probabilistic Symbolic Model Checker (PRISM) is, as its name implies, a probabilistic model checking tool that uses symbolic verification techniques. PRISM provides a modular specification language (based on the reactive modules language of [2]) in which to define a system. Henceforth, we use the term PRISM to describe both the model checking tool and the modelling language.

PRISM can generate a representation of a model from the high-level specification. Support is provided for reasoning over three types of models: DTMCs, MDPs and *continuous time Markov chains* (CTMCs). The first two have been discussed previously. We do not consider the latter.

Properties can also be specified in the PCTL (for DTMCs and MDPs) or CSL (for CTMCs) probabilistic temporal logics, which it can then automatically verify. Note that the definition of PCTL only allows a formula to be verified against a probability bound, but PRISM must calculate the probability of a formula being satisfied in order to compare it to the given bound. It is therefore straightforward to return the probability rather than a true or false answer. The PRISM model checker allows a property to be specified in this way. Note that, in the case of a MDP, it is only possible to return the minimum or maximum probability of a property being satisfied as it must be resolved across all possible non-deterministic choices.

For verification, PRISM provides explicit-state, symbolic and hybrid engines. The explicit-state engine employs a sparse matrix to store the transition matrix of a DTMC or transition function of a MDP. These structures generally contain a large number of zero entries and sparse matrices can exploit this redundancy.

An alternative to sparse matrices, employed by the symbolic verification engine, is the *Multi-Terminal Binary Decision Diagram* (MTBDD) data structure. MTBDDs are a generalisation of BDDs (described in Section 2.4.2). Whereas a BDD represents a function that maps boolean variables to a boolean value, an MTBDD represents a function that maps boolean variables to a value from an arbitrary domain (in this case the domain is always taken to be $\mathbb{R}$). Note that, as for a BDD, the variables of an MTBDD must be given an ordering and that this will significantly affect the the size of the MTBDD. An

MTBDD must also be reduced to its most compact form, using the same techniques as for a BDD, resulting in a canonical MTBDD.

Once an MTBDD or sparse matrix representation of a model (DTMC or MDP) is created, it can then be used for probabilistic model checking of PCTL properties. Qualitative properties (those that hold with probability 0 or 1) can be computed efficiently using BDDs.

Quantitative properties, on the other hand, require numerical calculation, which can be carried out using a number of different iterative techniques (e.g. Jacobi, Gauss-Seidel) [55]. Iterative methods work by maintaining a vector of solutions that initially contains some approximation to the actual result. A series of iterations takes place with a new solution vector being computed at each step, based on the old value of the vector and the matrix representing the model. The iterations normally terminate whenever the difference between the old and new solution vectors is less than some threshold value. Computation of the new vector mainly consists of a matrix-vector multiplication operation.

MTBDDs can be used to represent the matrix and the solution vector and efficient algorithms to perform the matrix-vector multiplication exist. However, in practice the numerical computation required to verify quantitative properties is often slow when performed on an MTBDD, due mainly to the growth of the MTBDD representation of the vector as the computation proceeds. Sparse matrices on the other hand store the vector in arrays, which remain constant in size, and so tend to perform much better than MTBDDs but have the disadvantage of having much greater memory requirements.

PRISM includes a hybrid engine that combines the MTBDD and sparse matrix approaches to provide a verification engine that is comparable in terms of speed to a sparse matrix implementation but also gives considerable savings in space. The hybrid technique uses an MTBDD to represent the model and a sparse matrix to represent the solution vector.

Note that PRISM also provides an option to apply fairness to the verification of MDPs based on the following definition, which is taken from [10].

Definition 3.4.1. *A path ω of an MDP $\mathcal{M}$ is* fair *if, for states s occurring infinitely often in ω, each choice $\mu \in Steps(s)$ is taken infinitely often. An adversary $A \in Adv_M$ is* fair *if, for any state s in $\mathcal{M}$, $Prob_s^A(\{\omega \in Path_s^A | \omega \text{ is fair}\}) = 1$.*

Note the difference between this definition of fairness and that used by SPIN which only provides a notion of fairness between *processes*.

For any MDP $\mathcal{M}$, let $\text{Adv}_{\mathcal{M}}^{fair} \subseteq \text{Adv}_{\mathcal{M}}$ denote the set of fair adversaries of $\mathcal{M}$. The fair satisfaction relation is denoted by $\models_{fair}$. The semantics for this are defined as for the satisfaction relation, $\models$, for all PCTL operators with the exception of $\mathcal{P}_{\bowtie}[\psi]$. The semantics for this are defined formally as follows,

$$s \models_{fair} \mathcal{P}_{\bowtie}[\psi] \Leftrightarrow Prob_s^A(\{\omega \in Path^A(s) | \omega \models \psi\}) \bowtie p \text{ for all } A \in \text{Adv}_{\mathcal{M}}^{fair},$$

where $\mathcal{M}$ is a MDP and s a state in $\mathcal{M}$.

3.4.1 The PRISM Modelling Language

The main features of the PRISM specification language are model type specification, process specification, synchronisation and probabilistic (and, in the case of MDPs, non-deterministic) choice. This section provides an outline of each of these, but note that a more thorough explanation can be found elsewhere in the literature [1].

The underlying model that is created from the code of a PRISM specification can be one of a DTMC, MDP or CTMC. To distinguish between these models in the specification it is necessary to include the keyword `probabilistic`, `nondeterministic` or `stochastic`, respectively. We consider only the two former model types.

Processes are defined using the `module` keyword followed by the name of the process. Multiple instantiations of any process can be declared by textual renaming. This is done by declaring the process using `module` as before followed by the process name (which must differ from the name of any other process). Each of the variables from the original process specification must then be renamed. As an example assume that we have a module, `module Site1` with variable `state1`. A new 'Site' process is created by writing:

```
module Site2=Site1 [state2=state1] endmodule.
```

Processes do not take any parameters and do not need to do so because all variables are visible globally. A process can have associated local variables which can either be declared as integers or booleans. For example,

```
an_int : [0..4] init 0;
```

declares a variable that can be an integer in the range zero to four, with initial value zero, whereas,

```
a_bool : bool init false;
```

declares a boolean variable initialised to false.

Note that it is also possible to declare global variables and constants, outside of a process specification. Global variables are declared in the same way as local variables but the declaration must be prefixed by the term `global`. Constants, like variables, can be integers or booleans but may also be declared as floating point doubles. As an example, `const int N = 10;` declares a global integer constant, `N`, with value ten.

In order for a process to update a variable, the conditions under which that variable are to be updated must first be satisfied. For example, given an integer variable `x` and a boolean `y` then,

```
[] (x=1 & y=true) -> (y'=false & x'=2);
```

updates `y` to false and `x` to two when `x` is one and `y` is true. Note the use of the dash (') notation to indicate the updated variable. It should also be pointed out that the variables referenced in the guard may be any program variable (including another processes local variables) but the variables to be updated must be global or local to that particular process.

In PRISM, probability is introduced in different ways depending on the type of Markov chain that is being used. Consider the case of DTMCs. Here probabilistic choice is introduced on the right hand side of the variable updates by using the '+' operator to indicate choice between a set of updates and then by labelling each separate update rule with a probability. For example, given `x` and `y` as above, given the statement,

```
[] (x=1 & y=true) -> 0.5 :(y'=false & x'=2) + 0.5 :(x'=0);
```

with equal probability, either an update is made to `y` and `x` is set to two or `x` is set to zero and `y` is unchanged (assuming that the guard condition holds). Note that the probabilities in a single update must sum to one.

In DTMCs if there is more than one update rule with a guard that evaluates to true

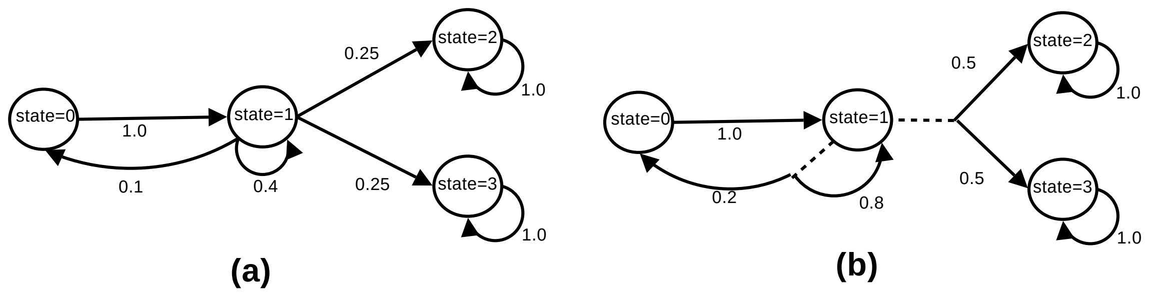


Figure 3.5: Comparison of a DTMC and a MDP

then the choice between these is made *probabilistically*, so that each update has an equal probability of being executed. In a MDP, however, such a choice is always made *non-deterministically*. To clarify, consider the PRISM specification given below.

```
module coin

 state : [0..3] init 0;

 [] state=0 -> 1.0 : state'=1;
 [] state=1 -> 0.8 : (state'=state) + 0.2 : (state'=0);
 [] state=1 -> 0.5 : (state'=2) + 0.5 : (state'=3);
 [] (state=2 | state=3) -> 1.0 : (state'=state);

endmodule
```

Note that we omit the declaration of the type of model from the specification. Suppose that we do so, using `probabilistic` or `nondeterministic`, defining a DTMC or MDP model respectively. These models are given in Figure 3.5. Figure 3.5 (a) shows the DTMC for the specification using the declaration `probabilistic` and Figure 3.5 (b) shows the MDP derived from specification using the declaration `nondeterministic`. Note that for the MDP model, the dashed lines indicate the nondeterministic transitions and the solid lines the probabilistic ones.

The important point to note is that, in the DTMC model, the outgoing transitions from the state in which `state=1` are all probabilistic, whereas in the MDP there are two non-deterministic transitions. This is as a result of the two statements in the specification which can both be executed if `state=1`. In the DTMC model the probabilities are all divided by two (the number of executable statements) to give transitions probabilities of $0.4, 0.1, 0.25$ and 0.25. On the other hand, in the specification of the MDP the two statements will be chosen non-deterministically.

One feature of the PRISM language that has not yet been discussed is *synchronisation*. In PRISM synchronisation between modules can be used as an alternative to global variables to enable processes to communicate (synchronously). In order to synchronise on some action, any update rules that are involved must be labelled with the same action name. This action can only then occur when each of the update statements are enabled, in which case the update in each statement is made simultaneously with probability given by the multiplication of the probabilities of the individual updates.

For example, consider two separate modules, one that includes the statement,

```
[action1] (x=0) -> 0.5 :  x'=1 + 0.5 :  x'=2;
```

and another with the statement,

```
[action1] (y=0) -> 0.5 :  y'=1 + 0.5 :  y'=2;.
```

Whenever `x` and `y` equal zero then `x` and `y` are updated so that: with probability 0.25 `x=y=1`; with probability 0.25 `x=1` and `y=2`; with probability 0.25 `x=2` and `y=1`; with probability 0.25 `x=y=2`.

In the case studies of randomised distributed protocols that we provide in later chapters we consider communication to be asynchronous. PRISM does not provide explicit structures for modelling asynchronous communication between components. However, it is relatively straightforward to define modules that act as communication buffers. In the case where we consider buffers of length one then we can in fact represent these with a global variable. In doing so note that we must be careful about the use of these buffers since ideally we want to restrict the type of operation that we can perform on them to reading and writing. This cannot be checked automatically: we must manually ensure that this restriction is maintained.

3.5 Probabilistic Relations

In the following sections we consider the bisimulation, simulation and stuttering equivalence relations for MDPs and the isomorphism relation for DTMCs. Note that the different relations preserve different probabilistic temporal logic properties between related structures. In particular bisimulation between MDPs preserves PCTL properties whereas stuttering equivalence preserves QLTL properties. This is an important point as it influences the probabilistic logic that we consider in the following chapters.

3.5.1 Strong Bisimulation and Simulation for MDPs

As for non-probabilistic structures, bisimulation and simulation can be defined for probabilistic models, though the definition must be lifted to probability distributions over states rather than single states. Bisimulation and simulation over MDPs preserves properties expressed in probabilistic temporal logic (e.g. PCTL). Note that we define the relations on a single structure rather than between a pair of models. However, it is straightforward to apply the definitions to two structures if we consider the disjoint union of the sets of states of the Markov models.

Strong Bisimulation A strong bisimulation is an equivalence relation that preserves the probability of transitions between equivalence classes. The formal definition of strong bisimulation for MDPs is given below. This is adapted from the definition given in [58] which defines strong bisimulation for probabilistic transition systems that are not state-labelled.

Definition 3.5.1. *Let $\mathcal{M} = (S, s_0, Act, Steps, L)$ be a MDP and R an equivalence relation on S. R is a* strong bisimulation *on $\mathcal{M}$ if, for s_1Rs_2:*

1. *$L(s_1) = L(s_2)$*
2. *For any $(a, \mu_1) \in Steps(s_1)$, there exists $(a, \mu_2) \in Steps(s_2)$ such that $\mu_1(C) = \mu_2(C)$ for all C in the set of equivalence classes S/R.*

For states s_1 and s_2 in some MDP, $\mathcal{M}$, s_1 is said to be strongly bisimilar to s_2, denoted by $s_1 \approx s_2$, if and only if there exists a strong bisimulation, R on $\mathcal{M}$, with s_1Rs_2.

Let $\mathcal{M} = (S, s_0, \mathit{Steps}, \mathit{Act}, L)$ and $\mathcal{M}' = (S', s_0', \mathit{Steps}', \mathit{Act}, L')$ be MDPs with sets of atomic propositions AP and AP' respectively, let $AP^* = AP \cap AP'$ and let $\uplus$ denote disjoint union. Define the *combination* of $\mathcal{M}$ and $\mathcal{M}'$ by $\mathcal{M}^* = (S \uplus S', \{s_0, s_0'\}, \mathit{Steps}^*, \mathit{Act}, L^*)$, where, for $s \in S \uplus S'$,

$$L^*(s) = \begin{cases} L(s) \cap AP^* & \text{if } s \in S \\ L'(s) \cap AP^* & \text{if } s \in S' \end{cases}$$

and, for $a \in \mathit{Act}$, $\mu \in \mathit{Dist}(S \uplus S')$, $(a, \mu) \in \mathit{Steps}^*(s)$ if and only if $s \in S$ and $(a, \mu) \in \mathit{Steps}(s)$ or $s \in S'$ and $(a, \mu) \in \mathit{Steps}'(s)$.

$\mathcal{M}$ is said to be strongly bisimilar to $\mathcal{M}'$, denoted by $\mathcal{M} \approx \mathcal{M}'$ if and only if there exists a strong bisimulation, R, on $\mathcal{M}^*$ and $s_0 R s_0'$.

We now give a logical characterisation of bisimulation in terms of PCTL, establishing which properties are preserved under the bisimulation relation. The following has been proved in [58].

Lemma 3.5.2. *Let $\mathcal{M}$ be a MDP with atomic propositions AP. If s_1, s_2 are states of $\mathcal{M}$ and ϕ is a PCTL property with propositions from AP, then,*

$$s_1 \approx s_2 \implies (s_1 \models \phi \iff s_2 \models \phi).$$

In other words two strongly bisimilar states satisfy the same PCTL properties. In addition, if $\mathcal{M} \approx \mathcal{M}'$, $\mathcal{M} \models \phi \iff \mathcal{M}' \models \phi$ for any PCTL property ϕ with propositions in AP.

Strong Simulation Below we define a simulation relation similar to that of [58]. In order to formalise a notion of strong simulation we first define a *weight function*.

Definition 3.5.3. *Let S, T be sets, $R \subseteq S \times T$ and $\mu \in$ Dist(S) ,$\nu \in$ Dist(T). A* weight function *for μ and ν with respect to R is a function $w : S \times T \to [0, 1]$ such that:*

- *$w(s, t) > 0 \Rightarrow sRt$,*
- *$\mu(s) = \sum_{t \in T} w(s, t)$ for any $s \in S$,*
- *$\nu(t) = \sum_{s \in S} w(s, t)$ for any $t \in T$.*

We write $\mu \sqsubseteq_R \nu$ (or simply $\mu \sqsubseteq \nu$ if R is clear from the context) if and only if there exists a weight function for μ and ν with respect to R.

Definition 3.5.4. *Let $\mathcal{M} = (S, s_0, Act, Steps, Act, L)$ and $\mathcal{M}' = (S', s'_0, Steps', Act, L')$ be MDPs with atomic propositions AP, AP' respectively such that $AP \subseteq AP'$. Let $R \subseteq S \times S'$. R is a* simulation *if and only if,*

- $s_0 R s'_0$,
- $\forall s \in S, s' \in S'$, *if* sRs' *then* $L'(s') \cap AP = L(s)$,
- $\forall s \in S, s' \in S'$, *if* sRs' *then for any* $(a, \mu') \in$ *Steps'*(s'), *there exists* $(a, \mu) \in$ *Steps*(s) *such that* $\mu \sqsubseteq_R \mu'$.

For states, $s \in S, s' \in S'$, we say that s *simulates* s', written $s \succeq s'$, if and only if there exists a simulation R, such that sRs'. Moreover, we say that $\mathcal{M}$ *simulates* $\mathcal{M}'$, denoted $\mathcal{M} \succeq \mathcal{M}'$ if and only if $s_0 \succeq s'_0$.

We subsequently give a logical characterisation of simulation with respect to PCTL. Strong simulation does not preserve all PCTL formulae, only PCTL-liveness formulae (see Definition 3.3.1). This result follows from a similar result [58] for probabilistic automata.

Lemma 3.5.5. *Let $\mathcal{M} = (S, s_0, Steps, Act, L)$, $\mathcal{M}' = (S', s'_0, Steps', Act', L')$ be MDPs with atomic propositions AP, AP' respectively such that $AP \subseteq AP'$. Let ϕ be a PCTL-liveness formula that only includes propositions from AP. If $s \succeq s'$ ($s \in S$, $s' \in S'$) then $s \models \phi \Rightarrow s' \models \phi$.*

3.5.2 Stuttering Equivalence

For any (finite or infinite) word v in some language $\mathbf{L}$, the stuttering removal operator, $\#$, applied to v replaces every maximal finite subsequence of identical elements by a single copy of this element. Two words v and w (that are both either finite or infinite) in $\mathbf{L}$ are said to be stuttering equivalent if and only if $\#v = \#w$ [49]. More detail in respect of stuttering equivalence is provided in [44].

We can apply the concept of stuttering equivalence to paths in MDPs if we consider the trace of each path as a word in the language given by sequences over the sets of atomic propositions. Formally, let $\mathcal{M}$ and $\mathcal{M}'$ be MDPs with atomic propositions AP and AP' respectively. An infinite (respectively finite) path ω of $\mathcal{M}$ is said to be *stuttering equivalent* to an infinite (respectively finite) path ω' of $\mathcal{M}'$ with respect to some common set of atomic propositions $AP'' \subseteq AP \cap AP'$ if and only if $\#trace^{AP''}(\gamma) = \#trace^{AP''}(\beta)$. We denote this by $\omega \simeq \omega'$.

We can extend stuttering equivalence of paths of MDPs to adversaries. To do so we need to define a *trace cylinder* over sequences of sets of atomic propositions [8].

Definition 3.5.6. *Let AP be a set of atomic propositions. The* trace cylinder

$$\mathcal{C}(l_0^+, l_1^+, \ldots, l_n^+)$$

(for $l_0, l_1, \ldots, l_n \in 2^{AP}$ pairwise distinct) is defined by

$$\mathcal{C}(l_0^+, l_1^+, \ldots, l_n^+) = \{t \in (2^{AP})^\omega) | t = l_0^{k_0}, l_1^{k_1}, \ldots, l_n^{k_n}, \ldots \textit{ for some } k_0, k_1, \ldots, k_n \geq 1\}.$$

where $l^k = \underbrace{l, l, \ldots, l}_{k}$ for $l \in 2^{AP}$ and $k \geq 1$.

For an adversary A of a MDP $\mathcal{M}$ over propositions AP with initial state s_0, and AP', a subset of the atomic propositions AP, by abuse of notation let,

$$Prob_{s_0}^A(\mathcal{C}(l_0^+, l_1^+, \ldots, l_n^+)) = Prob_{s_0}^A(\{\omega \in Path_{s_0}^A | trace^{AP'}(\omega) \in \mathcal{C}(l_0^+, l_1^+, \ldots, l_n^+)\}),$$

where $l_0, l_1, \ldots, l_n \in 2^{AP'}$.

Definition 3.5.7. *[8] A* stutter-invariant language *over a set of atomic propositions, AP, is a set $\mathbf{L} \subseteq (2^{AP})^\omega$ of infinite words over 2^{AP} such that, for any word $\theta \in \mathbf{L}$, all words θ' that are stutter equivalent to θ are also contained in $\mathbf{L}$. In order for $\mathbf{L}$ to be measurable it must also be an element of the σ-field generated by the trace-cylinders $\mathcal{C}(l_0^+, l_1^+, \ldots, l_n^+)$ where $l_0, l_1, \ldots, l_n$ are pairwise distinct subsets of AP.*

Given an adversary A for a MDP $\mathcal{M}$ with atomic propositions AP and a set of atomic propositions AP', a subset of AP and a state s, we abuse notation and henceforth let,

$$Prob_s^A(\mathbf{L}) = Prob_s^A(\{\alpha \in Path_s^A | trace^{AP'}(\alpha) \in \mathbf{L}\}),$$

denote the probability measure over the set of paths induced by A that start in state s and give a trace in a stutter-invariant language $\mathbf{L}$ over AP'.

Definition 3.5.8. *[8] Let $\mathcal{M} = (S, s_0, Steps, Act, L)$, $\mathcal{M}' = (S', s_0', Steps', Act', L')$, be MDPs with atomic propositions AP and AP' respectively. Two adversaries $A \in Adv_{\mathcal{M}}$, $A' \in Adv_{\mathcal{M}'}$ are probabilistic stuttering equivalent w.r.t. $AP'' \subseteq AP \cap AP'$ if and only if,*

$$Prob^{A}_{s_0}(\mathbf{L}) = Prob^{A'}_{s_0}(\mathbf{L}),$$

for all stutter-invariant measurable languages $\mathbf{L}$ over AP''.

By standard arguments of measure theory, it is sufficient, when determining probabilistic stuttering equivalence between adversaries, to consider only the trace cylinders over the sets of atomic propositions [8].

Proposition 3.5.9. *Let $\mathcal{M} = (S, s_0, Steps, Act, L)$ and $\mathcal{M}' = (S', s_0', Steps', Act', L')$, be two MDPs, with atomic propositions AP and AP' respectively. Two adversaries $A \in Adv_{\mathcal{M}}$, $A' \in Adv_{\mathcal{M}'}$ are probabilistic stuttering equivalent w.r.t. $AP'' \subseteq AP \cap AP'$ if and only if,*

$$Prob^{A}_{s_0}(\mathcal{C}(l_0^+, l_1^+, \ldots, l_n^+)) = Prob^{A'}_{s_0'}(\mathcal{C}(l_0^+, l_1^+, \ldots, l_n^+))$$

for all pairwise distinct $l_0, l_1, \ldots, l_n \in 2^{AP''}$ for $n \geq 0$.

Definition 3.5.10. *[8] Two MDPs, $\mathcal{M}$ and $\mathcal{M}'$, with atomic propositions, AP and AP', respectively, are said to be probabilistic stutter equivalent w.r.t. $AP'' \subseteq AP \cap AP'$ if and only if every adversary of $\mathcal{M}$ is probabilistic stuttering equivalent to some adversary of $\mathcal{M}'$ w.r.t AP'', and vice versa.*

In the remainder of this thesis, when it is clear from the context, we will use the term *stuttering equivalence* to refer to probabilistic stuttering equivalence between MDPs and adversaries *and* stuttering equivalence between paths. Furthermore, we overload the $\simeq$ operator and denote stuttering equivalence between two adversaries A and A' with respect to some set of atomic propositions, AP, by $A \simeq A'$ with respect to AP. Similarly, stuttering equivalent MDPs, $\mathcal{M}$ and $\mathcal{M}'$, with respect to AP, are denoted by $\mathcal{M} \simeq \mathcal{M}'$ with respect to AP.

Stuttering equivalent paths satisfy the same set of LTL$_{\setminus\mathcal{X}}$ properties [44]. It has also been shown that the languages induced by LTL properties are measurable [61]. Therefore, LTL$_{\setminus\mathcal{X}}$ properties induce stutter-invariant measurable languages. From Definition 3.5.8, it therefore follows that stuttering equivalent adversaries preserve the probability measure over paths satisfying LTL$_{\setminus\mathcal{X}}$ properties, a fact we state formally in Proposition 3.5.11.

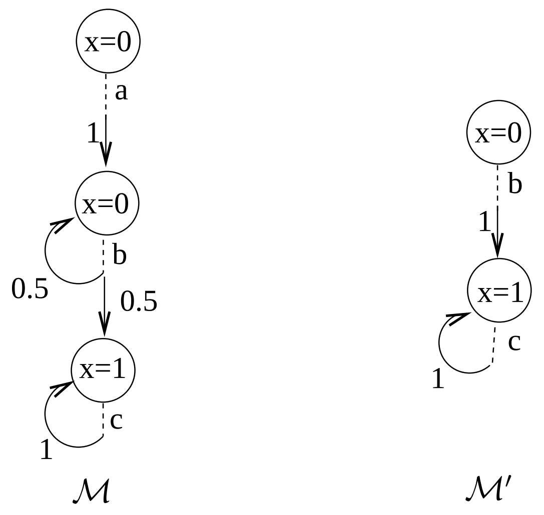


Figure 3.6: Two MDPs with stuttering equivalent adversaries

Proposition 3.5.11. *[8] If $\mathcal{M}$ and $\mathcal{M}'$ are MDPs with atomic propositions AP and AP', respectively, and adversaries A and A', respectively, then for any LTL$_{\backslash\mathcal{X}}$ path formula ψ with atomic propositions in $AP'' \subseteq AP \cap AP'$, if A is stuttering equivalent to A' w.r.t. AP'' then*

$$Prob_{s_0}^{A}(\{\omega \in Path_{s_0}^{A} | \omega \models \psi\}) = Prob_{s_0'}^{A'}(\{\omega' \in Path_{s_0'}^{A'} | \omega' \models \psi\}).$$

It follows that the supremum and infimum probability measures (with respect to the set of adversaries) over the set of paths, under an adversary, that satisfy some LTL$_{\backslash\mathcal{X}}$ property must be equal for two stuttering equivalent MDPs. Thus,

Lemma 3.5.12. *[8] If $\mathcal{M}$ and $\mathcal{M}'$ are MDPs with atomic propositions AP and AP', then for any QLTL$_{\backslash\mathcal{X}}$ property ϕ with atomic propositions in $AP'' \subseteq AP \cap AP'$, if $\mathcal{M}$ is stuttering equivalent to $\mathcal{M}'$ w.r.t. AP'' then,*

$$\mathcal{M} \models \phi \Leftrightarrow \mathcal{M}' \models \phi.$$

Example: To illustrate some of the concepts described thus far, in Figure 3.6 we give an example of two MDPs, $\mathcal{M}$ and $\mathcal{M}'$, with initial states s_0 and s_0' respectively. The MDP $\mathcal{M}$, has action set $Act = \{a, b, c\}$ and $\mathcal{M}'$ has action set $Act' = \{b, c\}$. Note that in both MDPs there is only one action enabled from every state and hence there is only one adversary associated with each MDP. Let these adversaries be A and A'. We show a fragment of the DTMCs obtained under these adversaries in Figure 3.7.

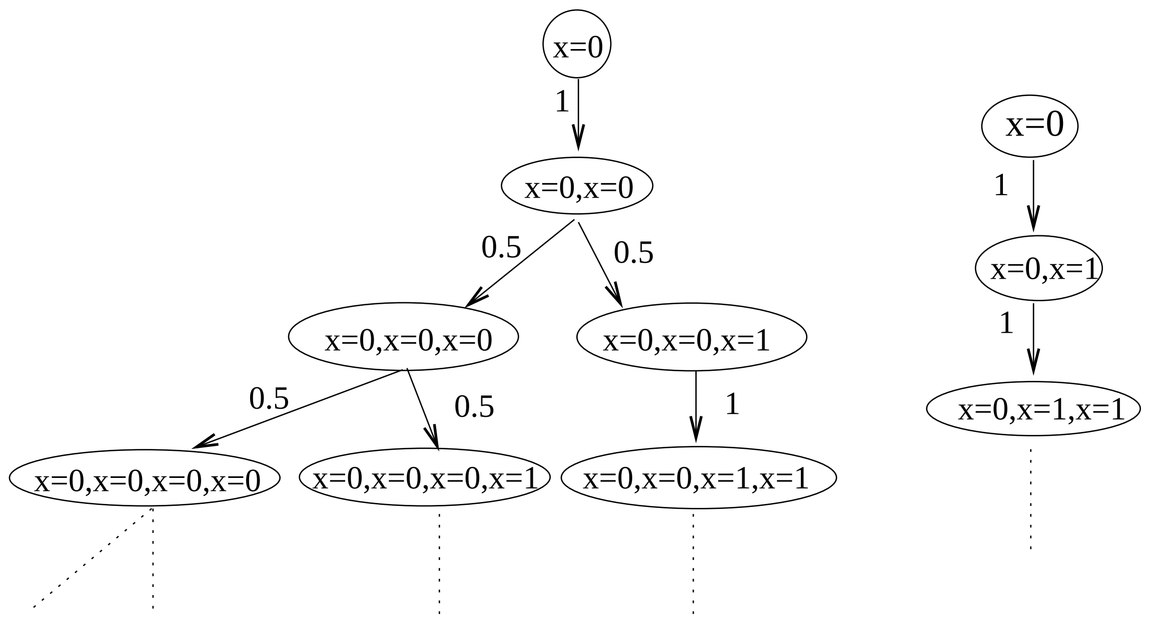


Figure 3.7: The DTMCs obtained under the (unique) adversaries of the MDPs in Figure 3.6

The adversaries A and A' are stuttering equivalent w.r.t. $AP = \{x = 0, x = 1\}$ since,

$$Prob^{A}_{s_0}(\mathcal{C}(\{x = 0\}^+)) = Prob^{A'}_{s'_0}(\mathcal{C}(\{x = 0\}^+)) = 1 \text{ and,}$$
$$Prob^{A}_{s_0}(\mathcal{C}(\{x = 0\}^+, \{x = 1\}^+)) = Prob^{A'}_{s'_0}(\mathcal{C}(\{x = 0\}^+, \{x = 1\}^+)) = 1,$$

and the probability measure over all other trace-cylinders is zero. Since A and A' are unique for $\mathcal{M}$ and $\mathcal{M}'$ respectively, $\mathcal{M} \simeq \mathcal{M}'$ w.r.t. AP.

Let ψ be the LTL$_{\setminus\mathcal{X}}$ path formula $true\,\mathcal{U}\; x = 1$. Then,

$$Prob^{A}_{s_0}(\{\omega \in Path^{A}_{s_0} | \omega \models \psi\}) = Prob^{A'}_{s'_0}(\{\omega' \in Path^{A'}_{s'_0} | \omega' \models \psi\}) = 1.$$

Thus, $\mathcal{M}$ and $\mathcal{M}'$ both satisfy the QLTL$_{\setminus\mathcal{X}}$ property $\mathcal{P}_{\geq 1}[\psi]$.

The proof of the following result is given in [56] for a more general case. For the sake of completeness we provide the proof for the special case we consider here.

Lemma 3.5.13. *Let $\mathcal{M} = (S, s_0, Steps, Act, L)$ and $\mathcal{M}' = (S', s'_0, Steps', Act', L')$ be MDPs with sets of atomic propositions AP and AP' respectively. Let A and A' be adversaries of $\mathcal{M}$ and $\mathcal{M}'$ respectively, and let $AP'' \subseteq AP \cap AP'$. Suppose that there exists cuts $D_0, D_1, \ldots$ with, for all $i \geq 0$, $D_i \in Cut(A')$, such that*

1. *$\forall i \geq 0$, $\forall \alpha \in D_{i+1}$, $\alpha \in D_i$ or for some $\beta \leq \alpha$, $\beta \in D_i$,*

2. *For every $\alpha \in Path^A_{fin}(s_0)$, $\lim_{i\to\infty} \sum_{\beta \in D_i, \alpha \leq \beta} \mathbf{P}(\beta) = Prob^A_{s_0}(\mathcal{C}(\alpha))$,*

3. *For each $i \geq 0$, define $\mu_i : cut^A_i \to [0,1]$, $\mu'_i : D_i \to [0,1]$ such that for $\alpha \in cut^A$, $\alpha' \in D_i$, $\mu_i(\alpha) = \mathbf{P}(\alpha)$, $\mu'_i(\alpha') = \mathbf{P}(\alpha')$. Then $\mu_i \sqsubseteq_R \mu'_i$ where for $\alpha \in cut^A_i$, $\alpha' \in D_i$, $R(\alpha, \alpha')$ iff $\alpha \simeq \alpha'$ w.r.t. AP'',*

then, A is stuttering equivalent to A'.

Proof. Let $l_0, l_1, \ldots, l_n \in AP''$ be pairwise distinct. Notice that,

$$Prob^A_{s_0}(\mathcal{C}(l_0^+, l_1^+, \ldots, l_n^+)) = \sum_{\alpha \in T^A_{l_0, l_1, \ldots, l_n}} Prob^A_{s_0}(\mathcal{C}(\alpha)),$$

where

$$T^A_{l_0,l_1,\ldots,l_n} = \{\alpha \in Path^A_{fin}(s_0) | trace^{AP''}(\alpha) \simeq l_0, l_1, \ldots, l_n \wedge \forall \beta < \alpha, trace^{AP''}(\beta) \not\simeq l_0, l_1, \ldots, l_n\}.$$

Alternatively,

$$Prob^A_{s_0}(\mathcal{C}(l_0^+, l_1^+, \ldots, l_n^+)) = \lim_{i\to\infty} \sum_{\alpha \in cut^A(i), \exists \beta \leq \alpha. \beta \in T^A_{l_0,l_1,\ldots,l_n}} Prob^A_{s_0}(\mathcal{C}(\alpha)).$$

Similarly,

$$Prob^{A'}_{s'_0}(\mathcal{C}(l_0^+, l_1^+, \ldots, l_n^+)) = \lim_{i\to\infty} \sum_{\alpha' \in D_i, \exists \beta' \leq \alpha'. \beta' \in T^{A'}_{l_0,l_1,\ldots,l_n}} Prob^{A'}_{s'_0}(\mathcal{C}(\alpha')).$$

Let $i \geq 0$ and let w be the weight function such that $\mu_i \sqsubseteq_R \mu'_i$. Then,

$$\begin{aligned}
& \sum_{\alpha \in cut^A(i), \exists \beta \leq \alpha. \beta \in T^A_{l_0,l_1,\ldots,l_n}} Prob^A_{s_0}(\mathcal{C}(\alpha)) \\
= & \sum_{\alpha \in cut^A(i), \exists \beta \leq \alpha. \beta \in T^A_{l_0,l_1,\ldots,l_n}} \mu_i(\alpha), \\
= & \sum_{\alpha \in cut^A(i), \exists \beta \leq \alpha. \beta \in T^A_{l_0,l_1,\ldots,l_n}} \sum_{\alpha' \in D_i} w(\alpha, \alpha'), \\
= & \sum_{\alpha' \in D_i, \exists \beta' \leq \alpha'. \beta' \in T^{A'}_{l_0,l_1,\ldots,l_n}} \sum_{\alpha \in cut^A(i)} w(\alpha, \alpha'), \\
= & \sum_{\alpha' \in D_i, \exists \beta' \leq \alpha'. \beta' \in T^{A'}_{l_0,l_1,\ldots,l_n}} \mu'_i(\alpha'), \\
= & \sum_{\alpha' \in D_i, \exists \beta' \leq \alpha'. \beta' \in T^{A'}_{l_0,l_1,\ldots,l_n}} Prob^{A'}_{s'_0}(\mathcal{C}(\alpha')).
\end{aligned}$$

Thus we have that,

$$Prob^A_{s_0}(\mathcal{C}(l^+_0, l^+_1, \ldots, l^+_n)) = Prob^{A'}_{s'_0}(\mathcal{C}(l^+_0, l^+_1, \ldots, l^+_n)).$$

□

Stuttering Equivalent Sets of Paths

We now extend our definition of stuttering equivalence between adversaries to a subset of paths under some adversary. It is necessary to consider this for our results in Chapter 6.

Definition 3.5.14. *Let $\mathcal{M} = (S, s_0, Steps, Act, L)$, $\mathcal{M}' = (S', s'_0, Steps', Act', L')$ be MDPs and let AP and AP' be the sets of atomic propositions over $\mathcal{M}$ and $\mathcal{M}'$ respectively. Given (randomised) adversaries $A \in Adv_{\mathcal{M}}(\in RAdv_{\mathcal{M}})$, $A' \in Adv_{\mathcal{M}'}(\in RAdv_{\mathcal{M}'})$ and a set of paths $\Pi \subseteq Path^A_{s_0}$ (measurable under $Prob^A_{s_0}$), then, Π is* stuttering equivalent *to A' with respect to $AP'' \subseteq AP \cap AP'$ if and only if,*

$$\frac{Prob^A_{s_0}(\{\omega \in Path^A_{s_0} | trace^{AP''}(\omega) \in \mathbf{L}\} \cap \Pi)}{Prob^A_{s_0}(\Pi)} = Prob^{A'}_{s'_0}(\mathbf{L})$$

for all stutter-invariant measurable languages $\mathbf{L}$.

If Π is stuttering equivalent to A we denote this by $\Pi \simeq A$. Arguing in a similar manner as for Proposition 3.5.11 we can establish Lemma 3.5.15.

Lemma 3.5.15. *Let $\mathcal{M} = (S, s_0, Steps, Act, L)$, $\mathcal{M}' = (S', s'_0, Steps', Act', L')$ be MDPs and let AP and AP' be the sets of atomic propositions over $\mathcal{M}$ and $\mathcal{M}'$ respectively. Given two (randomised) adversaries $A \in Adv_{\mathcal{M}}(\in RAdv_{\mathcal{M}})$, $A' \in Adv_{\mathcal{M}'}(\in RAdv_{\mathcal{M}'})$, let $\Pi \subseteq Path^A_{s_0}$ be a set of paths measurable under $Prob^A_{s_0}$ such that Π is* stuttering equivalent *to A' with respect to $AP'' \subseteq AP \cap AP'$. Then, for a LTL path formula ψ with atomic propositions in AP'',*

$$\frac{Prob^A_{s_0}(\{\omega \in Path^A_{s_0} | \omega \models \psi\} \cap \Pi)}{Prob^A_{s_0}(\Pi)} = Prob^{A'}_{s'_0}(\{\omega' \in Path^{A'}_{s'_0} | \omega' \models \psi\}).$$

3.5.3 Isomorphism

Isomorphic DTMCs must have exactly the same structural behaviour (up to the labelling of states). Definition 3.5.16 is taken from [26] (adapted from [42]).

Definition 3.5.16. *Let $\mathcal{D} = (S, s_0, \mathbf{P}, L)$ and $\mathcal{D}' = (S', s_0', \mathbf{P}', L')$ be DTMCs and let $\varsigma : S \to S'$ be a bijection. Suppose that for all $s, t \in S$, $\mathbf{P}(s,t) = \mathbf{P}'(\varsigma(s), \varsigma(t))$. Then ς is an isomorphism from $\mathcal{D}$ to $\mathcal{D}'$, and $\mathcal{D}$ and $\mathcal{D}'$ are said to be isomorphic.*

The following result is adapted from [26].

Lemma 3.5.17. *Let $\mathcal{D} = (S, s_0, \mathbf{P}, L)$ and $\mathcal{D}' = (S', s_0', \mathbf{P}', L')$ be DTMCs with atomic propositions AP and AP', respectively, and let $\Sigma : AP \to AP'$ be a bijection. For a QLTL formula ϕ with atomic propositions taken from AP, $\Sigma(\psi)$ is the QLTL formula obtained from ϕ by replacing every atomic proposition a in AP with $\Sigma(a)$. Let $\varsigma : S \to S'$ be an isomorphism from $\mathcal{D}$ to $\mathcal{D}'$ such that, for all $s \in S$ and $a \in AP$, $a \in L(s) \iff \Sigma(a) \in L'(\varsigma(s))$. Then for any QLTL formula ϕ with atomic propositions from AP and $s \in S$,*

$$\mathcal{D}, s \models \phi \iff \mathcal{D}', \varsigma(s) \models \Sigma(\phi).$$

3.5.4 Isomorphism between Adversaries

Isomorphic adversaries must have exactly the same structural behaviour (up to labelling of states). Definition 3.5.18 and Lemma 3.5.19 are adapted from [26].

Definition 3.5.18. *Let $\mathcal{M} = (S, s_0, Steps, Act, L)$ and $\mathcal{M}' = (S', s_0', Steps', Act', L')$ be MDPs with adversaries A and A' respectively. Let $\rho : Path^A_{fin}(s_0) \to Path^{A'}_{fin}(s_0')$ be a bijection with $\rho(s_0) = s_0'$. Suppose, for all $\alpha \in Path^A_{fin}(s_0)$, if $A(\alpha) = (a, \mu)$ and $\rho(\alpha \xrightarrow{(a,\mu)} t) = \alpha' \xrightarrow{(a',\mu')} t'$ then $A'(\alpha') = (a', \mu')$ and $\mu(t) = \mu'(t')$ for all t such that $\mu(t) > 0$. Then ς is an isomorphism from A to A', and A and A' are isomorphic (denoted $A = A'$).*

Lemma 3.5.19. *Let $\mathcal{M} = (S, s_0, Steps, Act, L)$ and $\mathcal{M}' = (S', s_0', Steps', Act', L')$ be MDPs with propositions AP and AP', respectively and let $\Sigma : AP \to AP'$ be a bijection. For LTL property ψ with propositions in AP, $\Sigma(\psi)$ is the LTL formula obtained from ψ by replacing every proposition a with $\Sigma(a)$. Let ς be an isomorphism between adversaries A (of $\mathcal{M}$) and A' (of $\mathcal{M}'$) such that, for all $\alpha \in Path^A_{fin}(s_0)$ and $a \in AP$, $a \in L(last(\alpha)) \iff \Sigma(a) \in L'(last(\varsigma(\alpha)))$. Then for any LTL formula ψ with propositions from AP, $Prob^A_{s_0}(\psi) = Prob^{A'}_{s_0'}(\Sigma(\psi))$.*

Let $\mathcal{M} = (S, s_0, Steps, Act, L)$ and $\mathcal{M}' = (S', s_0', Steps', Act', L')$ be MDPs and let A and A' be adversaries of $\mathcal{M}$ and $\mathcal{M}'$ respectively. For a measurable set of paths $\Pi \subseteq Path^A(s_0)$, if ς is an isomorphism between A and A' then we say Π is isomorphic to $\Pi' \subseteq Path^{A'}(s'_0)$ (denoted $\Pi = \Pi'$) if and only if $\Pi' = \varsigma(\Pi)$ where $\varsigma(\Pi) = \{\varsigma(\pi)|\pi \in \Pi\}$. If ς and Σ are as defined in Lemma 3.5.19 then it should be clear that for a LTL formula ψ,

$$Prob^{A}_{s_0}(\{\pi \in \Pi|\pi \models \psi) = Prob^{A'}_{s'_0}(\{\pi' \in \Pi'|\pi' \models \Sigma(\psi)).$$

Summary We have reviewed some aspects of the field of model checking. In particular we have defined Discrete Time Markov Chains and Markov Decision Processes, adversaries of MDPs, the PCTL and QLTL probabilistic temporal logics and the PRISM model checking tool. We have also defined the bisimulation, simulation, stuttering equivalence and isomorphism relations for Markov decision processes.

Chapter 4

Parameterised Verification of Non-degenerative Distributed Systems

Outline In this chapter we construct a network invariant for a simple token ring protocol using a methodology based on data abstraction that has previously been applied to a number of non-probabilistic distributed systems. We extend the definition of data abstraction to probabilistic models so that we can construct an invariant for a probabilistic variant of the token ring protocol. Using probabilistic and non-probabilistic relations we show that proving the invariants satisfy a property is equivalent to verifying the property is satisfied for the token ring protocol, for any size of ring.

4.1 Introduction

As discussed in Chapter 1, the construction of a finite-state network invariant is a commonly employed technique when verifying parameterised distributed systems. Although it has been shown that a finite-state invariant does not always exist [62], this is often a useful method, particularly when considering distributed systems that have a regular topology (e.g. a star or a ring). The method of construction can vary depending on the type of system under consideration and the formalism used to specify the system. One such method is by *abstraction* (initially described in [21]). For example, an abstraction mapping is defined on the states of a *concrete* model, and on a property. Application of the map results in an *abstract* (or *reduced*) model (and property). Proving that the abstract model satisfies any 'abstracted' property implies that the property holds in the concrete model [38].

Abstraction in the construction of a network invariant for a variety of systems has been considered in [17,49]. Finite-state network invariants are constructed for a fully connected email system, a client-server system and a leader election protocol for star topologies. The network invariants are specified in Promela by defining an *abstract* process that captures the behaviour of an arbitrary number of *concrete* processes, and composing this process in parallel with some fixed number of 'concrete' processes.

Rather than employing the invariant rule [62], a relationship is established between the underlying models of the invariant specifications and the underlying models of the fixed size system specifications. However, instead of showing a direct relationship, an intermediate step is employed, based on *data abstraction* (see Section 4.2) of the models of the fixed size system specifications. This simplifies the proof and provides a guide for the construction of the invariant. The data abstracted models are shown to satisfy any properties satisfied by the model of the invariant specification. Properties of the concrete model of a system of any size can then be inferred from properties of the invariant (which are established by model checking).

To illustrate the approach, we construct a network invariant for a simple counter token ring protocol (CTRP) described in Section 4.3. *Concrete* models for the token ring protocol are defined in terms of Promela specifications that are generated from a script program. The *invariant* model is also described by a Promela specification, as outlined in Section 4.4. The SPIN model checker is used to model check various LTL properties for the invariant. In section 4.5 we prove that any property that holds in the model of the invariant of the token ring system will hold for a concrete model of the CTRP with any number of processes.

We then proceed in Section 4.6 to introduce probabilistic choice to the model of the CTRP and specify concrete probabilistic models using PRISM (Section 4.8). In Section 4.10 we extend the parameterised verification proof employed for the CTRP to these probabilistic models. To do this we construct an invariant model from a PRISM specification and show (through the use of an adaptation of data abstraction to probabilistic models) that properties of the invariant hold for the concrete models.

4.2 Data Abstraction

As discussed in the previous section, in this chapter we extend the approach, based on data abstraction [18]. Data abstraction is employed to create an abstract model for models described by a set of variables, e.g. the models derived from a Promela specification.

Definition 4.2.1. *Let $X = \{x_0, x_2, \ldots, x_{n-1}\}$ be a set of variables where x_i has domain $D(x_i)$, $\forall i$, $0 \leq i < n$. We define the set of atomic propositions over X by $AP = \{x = d | x \in X, d \in D(x)\}$. The domain of X is defined as $D(X) = D(x_0) \times D(x_1) \times \ldots \times D(x_{n-1})$.*

Definition 4.2.2. *Let X be a set of variables with domain $D(X)$ and let $init(X)$ be the tuple of initial values for the variables in X. A (labelled) Kripke structure $\mathcal{K}$ over X is a tuple $(D(X), init(X), R, L)$, where L labels each state with atomic propositions from the set AP over X.*

To define an abstract model of a system, surjective mappings are defined from the domains of the variables that describe a model to a set of values in an abstract domain. In doing so, the set of possible values that the variables can take is restricted to a smaller set. This can then be used to define a *reduced Kripke structure*. This has the obvious advantage of reducing the size of the model, while preserving the behaviour of the system (see Lemma 4.2.5, below).

Definition 4.2.3. *Let $X = \{x_0, x_1, \ldots, x_{n-1}\}$ denote a set of variables such that each variable x_i ranges over a set $D(x_i)$. A set of abstract values $D'(X) = D'(x_0) \times D'(x_1) \times \ldots \times D'(x_{n-1})$ is called an abstract domain of X if there exist surjections $h_0, h_1, \ldots, h_{n-1}$ such that $h_i : D(x_i) \to D'(x_i)$ for all $0 \leq i < n$. If such surjections exist they induce a surjection $h : D \to D'$ defined by $h((x_0, x_1, \ldots, x_{n-1})) = (h_0(x_0), h_1(x_1), \ldots, h_{n-1}(x_{n-1}))$.*

Note that the states of the reduced Kripke structure are represented by sets of propositions whereas the states of the original structure are represented by a tuple of values. The labelling of the states are still given as sets of propositions, however.

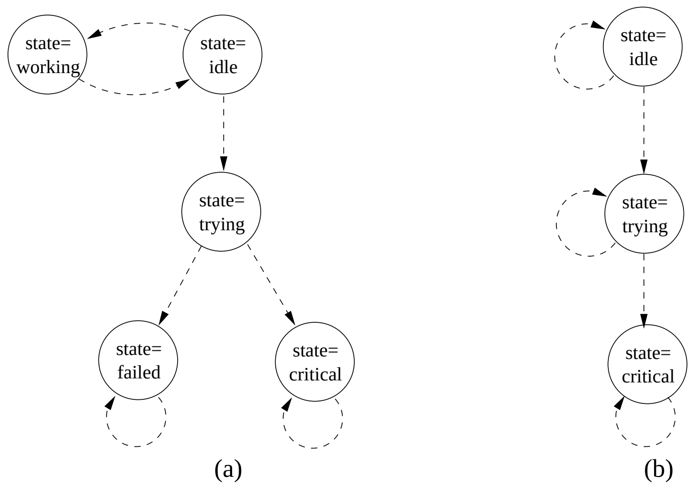


Figure 4.1: Example of a Kripke structure and a reduced structure under data abstraction

Definition 4.2.4. *[18] Let $\mathcal{K} = (S, R, s_0, L)$ be a Kripke structure over variable set X with the set of atomic propositions AP over X. If $D'(X)$ is an abstract domain of X and h the corresponding surjection from $D(X)$ to $D'(X)$ then h determines a set of abstract atomic propositions AP'. Let $\mathcal{K}'$ denote the structure identical to $\mathcal{K}$ but with the set of labels L' where L' labels each state with a set of abstract atomic propositions from AP'. Suppose that for all $s \in S$, $s \neq s_0$, $h(s_0) \neq h(s)$ then the structure $\mathcal{K}'$ can be collapsed into a reduced structure $\mathcal{K}_r = (S_r, R_r, s_r^0, L')$ where*

1. *$S_r = \{L'(s) | s \in S\}$, the set of abstract labels.*
2. *$AP_r = AP'$*
3. *As each $s_r \in S_r$ is a set of atomic propositions, $L'(s_r) = s_r$.*
4. *For $s_r, t_r \in S_r$, $R_r(s_r, t_r)$ if and only if there exists $s \in S$ and $t \in S$ such that $s_r = L'(s), t_r = L'(t)$ and $R(s,t)$.*

Intuitively, under data abstraction, groups of states of a model that were differentiated become equivalent. Equivalent states are then be represented as a single state in the reduced model. Any transitions to and from states in the original Kripke structure that have been data abstracted, must be preserved between the corresponding representative states in the reduced structure.

Example: Figure 4.1 shows how data abstraction is applied to a simple mutual exclusion protocol. In this case the model in Figure 4.1(a) consists of a single variable *state* which represents the state of a single process and can take any of the values in the domain $\{working, idle, trying, critical, failed\}$. Given the surjective mapping such that $working \mapsto working$, $idle \mapsto working$, $trying \mapsto trying$, $failed \mapsto trying$, $critical \mapsto critical$, the abstract domain of the variable *state* is given by the set $\{working, trying\ , critical\}$. The reduced structure is shown in Figure 4.1(b).

An important point to note from this example is that, in the reduced structure, the state labelled by the proposition $state = trying$ has a self-looping transition. This is because, in the original structure, the state with $state = trying$ can make a transition to $state = failed$. In the reduced structure $state = failed$ is data abstracted to $state = trying$ and so this state is 'merged' with the original state, $state = trying$. However, the transition in the original structure must be preserved and hence the self-loop must be included.

The following lemma is a restriction of a result in [20], which considered CTL* properties that only contain the **A** quantifier. The lemma states that any LTL properties that hold in a reduced (under data abstraction) structure of a system will also hold in the original [49].

Lemma 4.2.5. *If $\mathcal{K}$ is a Kripke structure with atomic propositions AP and $\mathcal{K}_r$ a reduced Kripke structure under h (as defined in Definition 4.2.4) with set of abstract atomic propositions AP' then for any LTL property ϕ with atomic propositions in AP, $\mathcal{K}_r \models h(\phi)$ implies that $\mathcal{K} \models \phi$ (where $h(\phi)$ maps the atomic propositions in ϕ to their abstract counterparts).*

Note that, in practice, we choose properties ϕ such that $h(\phi) = \phi$.

4.3 Specifying the Counter Token Ring Protocol (CTRP)

In this section we consider a counter token ring protocol (CTRP), in which processes pass a single token around a ring in one direction. If a process is ready to perform some operation (e.g. to transmit a message or to access some shared resource), it waits until it receives the token, performs its operation and then passes the token on. We include in the token a counter. Each time a process acquires the token and performs some operation, the counter is decremented by one. Once the counter reaches zero, the protocol terminates.

The counter could be used, for example, as a means to limit the number of operations that can be performed (see for example [59]).

We specify models of the CTRP using Promela; specifications for any fixed number of processes can be generated from a script program. Note that, in modelling the protocol, we use an arbiter process to decide when the processes should perform some operation (and therefore decrement the counter). An example of the Promela code for the *concrete specification* of the CTRP with four processes and one arbiter is given in Appendix A. We informally outline the behaviour of each process, below.

A process definition is given for a *process* and for the *arbiter*. The *arbiter* updates one global boolean variable, *work*. Once the *arbiter* has changed the value of *work*, it must wait for a *process* to set the global boolean variable *done* to the appropriate value so that *work* can be changed again. For example, if *work* is *true* then once *done* is set to *true*, *arbiter* can set *work* to *false*. Note that the *arbiter* process is always at the location labelled *change*, and therefore the program counter for *arbiter* can be considered a constant and thus ignored.

A *process* has three parameters: an *in* channel, an *out* channel and an index, *id*. Several *process* components are instantiated, with the *in* channel of each *process* being the *out* channel of its left 'neighbour' and the *id* of each one being unique. Note that channels are stored in a global array, *ch*, with each element being indexed by the id of the *process* which has that channel as its *in* channel. If a channel holds value k we denote this by $[k]$; if a channel is empty we denote this by [].

Initially, each *process* is at the location labelled *idle* and it polls its *in* channel for receipt of a counter token. If the channel is not empty, its value is read and stored in *token*, a local integer variable, *process* moves to the location labelled *rcvd* and a check is made to establish whether *work* is *true* or *false*. If *work* is *false* then *process* sends the value of *token* to its neighbour via its *out* channel and returns to *idle*. If *work* is *true*, *token* is decremented and a check is made to see if *token* equals zero. If *token==0*, then *end* is set to true and the process moves to *finish*. Otherwise *process* sets *done* to *true*, moves to location label *pass* and waits for *work* to be set to *false* before sending the new value of *token* to its neighbour and moving to label *idle*.

The specification also includes a number of variables which are only used for verification purposes. For example, each *process* has a local variable *possess*, which is set to *true* when the *process* receives the token and is reset to *false* when *token* is passed on. This can then be used to verify that no two processes ever have the token at the same time. Another local *process* variable, *working* is set to true whenever *token* is decremented and is reset to false when *token* has been passed on. This can be used, for example, to verify that an operation *can* be performed by *process*. Also, the global integer variable *finished* is incremented by one whenever a process sets the token value to zero.

At initialisation, $ch[0]$ is set to some positive non-zero value, all other channels are initialised empty and *work* and *done* are set to *false*. At this point, the processes must also be instantiated: a fixed number of *process*es are run in parallel with one *arbiter* process.

For $N > 1$ and $c \geq 1$, we let $\mathcal{S}(N, c)$ denote the specification:

```
init{
    atomic{
        run process(ch[0],ch[1],0);
        run process(ch[1],ch[2],1);
        ⋮
        run process(ch[N],ch[0],N);
        run arbiter();
        ch[0]!c; // always start with process 0
    }
}//init
```

Note that there are two parameters to our verification problem: the number of processes and the initial value of the counter. However, in constructing an invariant we only consider the former. The latter remains a parameter for the invariant.

To distinguish between the local variables of each *process*, we use var_i to denote the local variable *var* associated with the instantiation of *process* with *id* equal i. Similarly we refer to $process_i$.

4.3.1 Properties of the CTRP

A description of the properties of the CTRP that capture correct behaviour of the system are given below. The properties are formalised using LTL and have been verified using

SPIN for two, three, four and five processes (and one *arbiter* process) and the initial value of the counter between one and ten.

Safety

S1. Only one process at a time can hold the token.

$$\begin{aligned}&\Box((!possess_0\&\&!possess_1\&\&\ldots\&\&!possess_{N-1})\\ &||(possess_0\&\&!possess_1\&\&!possess_2\&\&\ldots\&\&!possess_{N-1})\\ &||(!possess_0\&\&possess_1\&\&!possess_2\&\&!possess_3\&\&\ldots\&\&!possess_{N-1})\\ &\vdots\\ &||(!possess_0\&\&!possess_1\&\&\ldots\&\&!possess_{N-2}\&\&possess_{N-1}))\end{aligned}$$

S2. At most one process can finish.

- $\Box(finished == 0||finished == 1)$

Liveness

L1. Eventually a process will finish.

- $(finished == 0)\mathbf{U}(finished == 1)$

L2. If process i (for each $0 \leq i \leq N-1$) receives the token it will always eventually pass the token to its neighbour, unless the protocol has terminated.

- $\Box(possess_i \rightarrow (\Diamond(!possess_i||end)))$

L3. Eventually the arbiter will schedule an operation.

- $\Diamond(work)$

L4. An operation can be performed infinitely often.

- $\Box\Diamond(work)$

L5. Once finished, every process must have a token with zero value.

- $\Diamond(end\&\&(token_0 == 0)\&\&(token_1 == 0)\&\&\ldots\&\&(token_{N-1} == 0))$

L6. Process i (for each $0 \leq i \leq N-1$) can perform an operation.

- $\Box(working_i)$ (show to be false)

L7. It is possible for process i (for each $0 \leq i \leq N-1$) to never perform an operation.

- $\Diamond(working_i)$ (show to be false)

L8. Eventually the protocol will end and will not restart.

- $\Diamond\Box(end)$

It should be observed that under normal circumstances properties **L1, L3, L4, L5** and **L8** do not hold because there exist paths along which the *arbiter* process never executes, so that *work* is *false* forever and the counter never decremented. As discussed in Section 2.4.1, SPIN provides an option to apply *weak fairness* during verification to ensure that any process that has an enabled transition will eventually execute it. In this case, by applying weak fairness during verification of properties **L1, L3, L4, L5** and **L8**, the *arbiter* process must eventually be executed, so that *work* is always eventually set to *true*, hence these properties hold. Note also that **L7** will hold without any fairness constraints, but weak fairness should still be applied to avoid searching infinite paths for which the property would (trivially) hold.

4.4 Specification of an Invariant for the CTRP

To describe an invariant model that captures the behaviour of every concrete model we specify in Promela an *abstract_process* that captures the behaviour of an arbitrary number of *process* processes. Intuitively, this process behaves the same as a *process*: it can receive the token from its neighbour and then either pass the token on or decrement the token and then either terminate or pass the token on. However, when passing the token, the *abstract_process* can non-deterministically choose to either send it to its neighbour or *send the token back to itself.* In this way the *abstract_process* behaves like an arbitrary number of *process*es.

The Promela code for the *invariant specification* of the CTRP with two concrete *process* components, an *arbiter* and an *abstract_process* is provided in Appendix B. Note that we can run any number of *process* components in parallel with the *abstract_process.* We assume that there are m such processes. We describe below the operation of *abstract_process.*

The *abstract_process* has the same local parameters as for a *process*, with *id* always being given the highest possible value (m), *out* aways being channel *ch[0]* and *in* always being *ch[m]*. As for *process*, in location *idle*, *abstract_process* will accept the token when it has been sent on its *in* channel, move to location *rcvd* and then check whether *work* is *true* or *false*.

However, once it has determined the status of *work*, the behaviour of *abstract_process* deviates from that of *process*. If *work* is *false*, it can still send the token to its neighbour but it may also non-deterministically choose to send the token to itself via its *in* channel, before returning to the *idle* location. In the latter case, execution will then continue with *abstract_process* reading the token from channel *in* and moving to *rcvd*. Note that if *work* is *false* forever, then it is possible for *abstract_process* to do this an infinite number of times.

On the other hand, if *work* is *true*, *abstract_process* must decrement *token*. Either *token* is zero, *end* is set to true, *finished* is incremented by one and execution jumps to *finish* or *token* is non-zero and *done* is set to *true*. In the latter case, execution then blocks at the location labelled *pass* until *work* is *false*, after which *done* is set to *false*. The *abstract_process* can then non-deterministically choose to send *token* to its neighbour or to itself before moving to location *idle*.

It should be apparent from the above description of *abstract_process* that the *possess* and *working* variables are not included in the process specification. This is a result of the data abstraction step which will be described in Section 4.5.

As described above, *abstract_process* is always given the highest *id* value (m say). It receives *token* from $process_{m-1}$ and passes it to $process_0$ (or to itself). Hence, the topology (of the ring) consists of m neighbouring concrete processes with a single *abstract_process* between $process_0$ and $process_{m-1}$ (see Figure 4.2). It should be noted, however, that it is also possible to define alternative topologies where the concrete processes may not be adjacent e.g. where an *abstract_process* is inserted between all neighbouring concrete processes.

Note that, as discussed earlier, the initial value of the counter remains a parameter in the invariant specification, and the number of concrete processes is also a parameter. Therefore, rather than specifying a single invariant we are in fact describing a parameterised

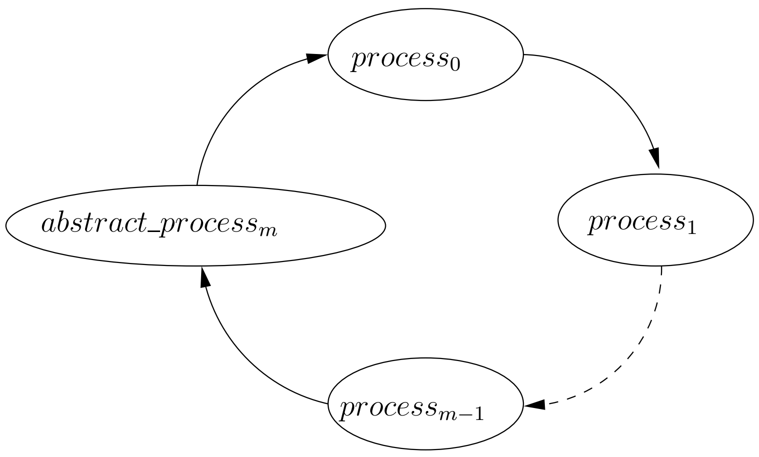


Figure 4.2: Topology for the abstract token ring specification (excluding the arbiter process) with m concrete processes

family of invariants. For $m > 1$ and $c > 1$, we let $\mathcal{I}(m, c)$ denote the invariant specification:

```
init{
    atomic{
        run process(ch[0],ch[1],0);
        run process(ch[1],ch[2],1);
        ⋮
        run process(ch[m − 1],ch[m],m − 1);
        run abstract_process(ch[m],ch[0],m);
        run arbiter();
        ch[0]!c; // always start with process 0
    }
}//init
```

where each *process* and *arbiter* are as defined previously and *abstract_process* is as described above and defined in Appendix B.

4.4.1 Properties of the Invariant

A number of LTL properties were described in Section 4.3.1 for the concrete Promela specification of the CTRP. Some LTL properties for the invariant specification are given below, again divided into the classes of safety and liveness. It is important to note that some of the properties below are different from those given for the concrete specification. As we will see in the following sections, the abstraction of the concrete models inhibits the set of variables that we can refer to when specifying our properties. Specifically we cannot make reference to any local variable of an abstract process in any property of the invariant that we wish to verify.

Safety

IS1. Only one *concrete* process at a time can hold the token.

$$\begin{aligned}&\Box((!possess_0\&\&!possess_1\&\&\ldots\&\&!possess_{m-1})\\&||(possess_0\&\&!possess_1\&\&!possess_2\&\&\ldots\&\&!possess_{m-1})\\&||(!possess_0\&\&possess_1\&\&!possess_2\&\&!possess_3\&\&\ldots\&\&!possess_{m-1})\\&\vdots\\&||(!possess_0\&\&!possess_1\&\&\ldots\&\&!possess_{m-2}\&\&possess_{m-1}))\end{aligned}$$

IS2. No more than one process can finish.

$$\Box(finished == 0||finished == 1)$$

Liveness

IL1. Eventually one process will finish.

$$((finished == 0)\mathbf{U}(finished == 1))$$

IL2. If concrete process i (for each $0 \leq i \leq m-1$) receives the token it will always eventually pass the token to its neighbour, unless the process has finished.

$$\Box(possess_i \rightarrow (\Diamond(!possess_i||end)))$$

IL3. Eventually the arbiter will schedule an operation.

$$\Diamond(work)$$

IL4. An operation can be performed infinitely often.

$$\Box\Diamond(work)$$

IL5. Once finished every concrete process must have a zero token.

$$\Diamond(end\&\&(token_0 == 0)\&\&(token_1 == 0)\&\&\ldots(token_{m-1} = 0))$$

IL6. Each process can perform an operation. For each $0 \leq i \leq m-1$,

$\Box !working_i$ (show to be false)

IL7. It is possible for a process to never perform an operation. For each $0 \leq i \leq m-1$,

$\Diamond(working_i)$ (show to be false)

IL8. Eventually the protocol will end and will not restart.

$$\Diamond\Box(end)$$

The properties **IS1**, **IS2**, **IL2**, **IL3**, **IL4**, **IL6** and **IL7** have been verified against the invariant specification of the CTRP with two *process* components and one *abstract_process* process, and with the initial value of the counter between one and ten. Note, though, that weak fairness had to be applied to verify properties **IL3** and **IL4**.

However, the remaining properties (**IL1, IL5**) do not hold even under the application of weak fairness. This is due to the introduction of cycles into the *abstract_process* as described in Section 4.4: intuitively it is possible for the *abstract_process* to pass the token to itself forever. We return to this point at the end of the chapter.

It should also be pointed out that properties **IS1**, **IL2**, **IL6** and **IL7** may not refer to the *possess* or *working* variables associated with the abstract *process* components. This is an example of the restriction that is given later in Theorem 4.5.1, which states that a property can only hold under abstraction if it does not index any abstract process components. This stipulation is necessary, and limits the type of property that can be verified. For example, it should be apparent that, in the abstract specification, **IS1** only shows that the concrete processes will never possess the token simultaneously, and does not give any information on whether any abstract process can hold the token at the same time as another process.

4.5 Parameterised Verification Proof for the CTRP

We prove that the model of the invariant specification captures the behaviour of the models of the concrete specifications i.e. we show that if any property (satisfying a given set of restrictions) holds in the invariant model then it will hold in the concrete model for any number of processes. We do this using a proof similar to those of [17, 49]. We give this proof in some detail here since it provides a basis for our proof for the probabilistic case in Section 4.6.

For $N > 1$ and $c > 1$, given a CTRP specification, $\mathcal{S}(N, c)$, let the *concrete* (Kripke structure) model of the CTRP be $\mathcal{M}_N$. Given an invariant specification $\mathcal{I}(m, c)$, for $m > 1$, let $\mathcal{M}_I^m$ be defined as the *invariant* model of the CTRP.

The processes with *id* between 0 and $m-1$ in both the invariant and concrete specifications will be referred to as the *concrete* processes while those with *id* between m and $N-1$ in the concrete specification will be described as the *abstract* processes.

We aim to prove the following:

Theorem 4.5.1. *Let $m > 1$ and let ϕ be an LTL property that does not contain any index greater than or equal to m in its atomic propositions. Then, for all $N > m$, $c \geq 1$, if $\mathcal{M}_I^m \models \phi$ then $\mathcal{M}_N \models \phi$.*

Proof Overview The proof is in two stages: creation of a reduced model from $\mathcal{M}_N$ (for each N) by data abstraction, followed by definition of a simulation relation between each reduced model and $\mathcal{M}_I^m$.

The construction of a reduced set of models proceeds as follows. For each *concrete* model $\mathcal{M}_N$ of the system with a fixed number of $N > m$ *process*es, and property ϕ, the processes are divided into *concrete* processes $p_0, p_1, \ldots, p_{m-1}$ and *abstract* processes $p_m, p_{m+1}, \ldots, p_{N-1}$. We assume property, ϕ, only refers to the concrete processes (i.e. the atomic propositions in ϕ include only the local and channel variables with index $0 \leq i < m$) and not to the abstract processes.

An abstract model $\mathcal{M}_N^m$ is constructed by *data abstraction* on the variables of the abstract processes (and possibly some of the variables of the concrete processes). From Definitions 4.2.3 and 4.2.4, this is done by defining a set of surjections that map the values of the variables in $\mathcal{M}_N$ to values in an abstract domain (in this case we assume the abstract domain is a subset of the concrete domain). From Lemma 4.2.5, any properties that hold in a reduced structure of a system will also hold in the original, and so, for all $N > m$, if $M_N^m \models \phi$ for some suitable property ϕ, then $\mathcal{M}_N \models \phi$.

The *invariant* model $\mathcal{M}_I^m$ is specified in such a way that it captures the behaviour of an arbitrary number of the abstract processes. For each $N > m$, we establish a simulation relation between $\mathcal{M}_N^m$ and $\mathcal{M}_I^m$, thereby showing that the latter simulates the former. From Lemma 2.5.4, any property that holds in a structure will also hold in any structure that it simulates. So if $\mathcal{M}_I^m$ simulates $\mathcal{M}_N^m$ and $\mathcal{M}_I^m \models \phi$ for a suitable property ϕ, then $\mathcal{M}_N^m \models \phi$. It then follows that $\mathcal{M}_N \models \phi$, for any value of N.

4.5.1 Creating Reduced Models by Data Abstraction

Let $m > 1$, $N > 1$. A requirement of the proof is to define a reduced structure based on the concrete model, by data abstraction. This is done by defining a set of surjections that map the values from the domain of variables of every concrete model, $\mathcal{M}_N$, to values in an abstract domain.

From the specification $\mathcal{S}(N)$, we know that the set of variables in the concrete model $\mathcal{M}_N$ is given by $\mathbf{X} = X_G \cup X_V \cup X_C$, where $X_G = \{\mathit{ready}, \mathit{work}, \mathit{end}, \mathit{finished}\}$ is the set of global variables, $X_V = \{\overline{\mathit{loc}}, \overline{\mathit{token}}, \overline{\mathit{possess}}, \overline{\mathit{working}}\}$ is the set of all local variables and $X_C = \{\overline{\mathit{ch}}\}$ is the set of channel variables with,

$$\begin{aligned}
\overline{\mathit{loc}} &= (\mathit{loc}_0, \ldots, \mathit{loc}_{N-1}) \\
\overline{\mathit{token}} &= (\mathit{token}_0, \mathit{token}_1, \ldots, \mathit{token}_{N-1}) \\
\overline{\mathit{possess}} &= (\mathit{possess}_0, \ldots, \mathit{possess}_{N-1}) \\
\overline{\mathit{working}} &= (\mathit{working}_0, \ldots, \mathit{working}_{N-1}) \\
\overline{\mathit{ch}} &= (\mathit{ch}[0], \mathit{ch}[1], \ldots, \mathit{ch}[N-1]).
\end{aligned}$$

Note the inclusion of the local variables loc_i. These represent the program counter variables for each process. The only execution points that a *process* can visit are those labelled by *idle*, *rcvd*, *pass* and *finish* in the specification and so, below, we give the domain for each of the loc_i variables as this set of labels.

For every variable in X_G, the surjective mapping is the identity map on the respective domains. The domains of X_V and X_C are given by,

$$
\begin{aligned}
&D(\overline{loc}) = D(loc_0) \times D(loc_1) \times \ldots \times D(loc_{N-1}),\\
&D(\overline{token}) = D(token_0) \times D(token_1) \times \ldots \times D(token_{N-1}),\\
&D(\overline{possess}) = D(possess_0) \times D(possess_1) \times \ldots \times D(possess_{N-1}),\\
&D(\overline{working}) = D(working_0) \times D(working_1) \times \ldots \times D(working_{N-1}),\\
&D(\overline{ch}) = D(ch[0]) \times D(ch[1]) \times \ldots \times D(ch[N-1]),
\end{aligned}
$$

where $\forall i$, $0 \leq i \leq N-1$,

$$
\begin{aligned}
&D(loc_i) = \{idle, rcvd, pass, finish\},\\
&D(token_i) = \{0, 1, \ldots, c\},\\
&D(possess_i) = D(working_i) = \{\text{true}, \text{false}\},\\
&D(ch[i]) = \{[], [1], \ldots, [c]\}.
\end{aligned}
$$

For $m > 1$, $N > m$ the abstract domains are given below. Note that we consider the *loc* and *token* variables, associated with the abstract processes, differently from the others. We consider a tuple of these values. This enables the variables to be data abstracted to a single value, which is necessary since we need to include a *loc* and *possess* variable in the invariant specification. The local variable, *possess*, for example, does not need to be considered in this way since we do not refer to it in the invariant specification.

$$
\begin{aligned}
&D'(\overline{loc}) = D'(loc_0) \times D'(loc_1) \times \ldots \times D'((loc_m, loc_{m+1}, \ldots, loc_{N-1})),\\
&D'(\overline{token}) = D'(token_0) \times D'(token_1) \times \ldots \times D'((token_m, \ldots, token_{N-1})),\\
&D'(\overline{possess}) = D'(possess_0) \times D'(possess_1) \times \ldots \times D'(possess_{N-1}),\\
&D'(\overline{working}) = D'(working_0) \times D'(working_1) \times \ldots \times D'(working_{N-1})),\\
&D'(\overline{ch}) = D'(ch[0]) \times D'(ch[1]) \times \ldots \times D'((ch[m], \ldots, ch[N-1])),
\end{aligned}
$$

where, $\forall i$, $0 \leq i \leq m-1$,

$$
\begin{aligned}
&D'(loc_i) = \{idle, rcvd, pass, finish\},\\
&D'(token_i) = \{0, 1, \ldots, c\},\\
&D'(possess_i) = D'(working_i) = \{true, false\},\\
&D'(ch[i]) = \{[], [1], \ldots, [c]\},
\end{aligned}
$$

and $\forall i$, $m \leq i \leq N-1$,

$$D'(possess_i) = D'(working_i) = \{true\},$$

and,

$$\begin{aligned}
&D'((loc_m, loc_{m+1}, \ldots, loc_{N-1})) = \{idle, rcvd, pass, finish\},\\
&D'((token_m, token_{m+1}, \ldots, token_{N-1})) = \{0, 1, \ldots, c\},\\
&D'((ch[m], ch[m+1], \ldots, ch[N-1])) = \{[], [1], \ldots, [c]\}.
\end{aligned}$$

Then for all $0 \leq i \leq N-1$, the corresponding surjections can be defined as follows

$$\begin{aligned}
&d : D(\overline{loc}) \rightarrow D'(\overline{loc})\\
&e : D(\overline{token}) \rightarrow D'(\overline{token})\\
&f : D(\overline{possess}) \rightarrow D'(\overline{possess})\\
&g : D(\overline{working}) \rightarrow D'(\overline{working})\\
&h : D(\overline{ch}) \rightarrow D'(\overline{ch})
\end{aligned}$$

$$\begin{aligned}
&d(\overline{loc}) = (d_0(loc_0), \ldots, d_{m-1}(loc_{m-1}), d_m((loc_m, \ldots, loc_{N-1})))\\
&e(\overline{token}) = (e_0(token_0), \ldots, e_{m-1}(token_{m-1}), e_m((token_m, \ldots, token_{N-1}))),\\
&f(\overline{possess}) = (f_0(possess_0), f_1(possess_1), \ldots, f_{N-1}(possess_{N-1})),\\
&g(\overline{working}) = (g_0(working_0), g_1(working_1), \ldots, g_{N-1}(working_{N-1})),\\
&h(\overline{ch}) = (h_0(ch[0]), \ldots, h_{m-1}(ch[m-1]), h_m((ch[m], \ldots, ch[N-1]))),
\end{aligned}$$

where $\forall i$, $0 \leq i \leq N-1$,

$$\begin{aligned}
&f_i : D(possess_i) \rightarrow D'(possess_i)\\
&g_i : D(working_i) \rightarrow D'(working_i)
\end{aligned}$$

and $\forall i$, $0 \leq i \leq m-1$,

$$\begin{aligned}
&d_i : D(loc_i) \rightarrow D'(loc_i)\\
&e_i : D(token_i) \rightarrow D'(token_i)\\
&h_i : D(ch[i]) \rightarrow D'(ch[i])
\end{aligned}$$

and,

$$\begin{aligned}
&d_m : D((loc_m, \ldots, loc_{N-1})) \rightarrow D'((loc_m, \ldots, loc_{N-1}))\\
&e_m : D((token_m, token_{m+1}, \ldots, token_{N-1})) \rightarrow D'((token_m, token_{m+1}, \ldots, token_{N-1}))\\
&h_m : D((ch[m], \ldots, ch[N-1])) \rightarrow D'((ch[m], \ldots, ch[N-1]))
\end{aligned}$$

For all i, $0 \leq i \leq m-1$, d_i, e_i, f_i, g_i and h_i are just the identity mappings on the appropriate domains. For all i, $m \leq i \leq N-1$, $f_i(\mathit{possess}_i) = g_i(\mathit{working}_i) = \mathit{false}$. Also,

$$d_m((\mathit{loc}_m, \mathit{loc}_{m+1}, \ldots, \mathit{loc}_{N-1})) = \begin{cases} \mathit{loc}_j & \text{if } \exists m \leq j \leq N-1.\mathit{loc}_j \neq \mathit{idle} \\ & \quad \wedge \forall m \leq i \neq j \leq N-1, \mathit{loc}_i = \mathit{idle} \\ \mathit{idle} & \text{otherwise} \end{cases}$$

$$e_m((\mathit{token}_m, \mathit{token}_{m+1}, \ldots, \mathit{token}_{N-1})) = \begin{cases} \mathit{token}_j & \text{if } \exists m \leq j \leq N-1.\mathit{token}_j > 0 \\ & \quad \wedge \forall m \leq i \neq j \leq N-1, \mathit{token}_i = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$h_m((\mathit{ch}[m], \mathit{ch}[m+1], \ldots, \mathit{ch}[N-1])) = \begin{cases} [t] & \text{if } \exists m \leq j \leq N-1.\mathit{ch}[j] = [t] \\ & \quad \wedge \forall m \leq i \neq j \leq N-1, \mathit{ch}[i] = [] \\ [] & \text{otherwise} \end{cases}$$

The reduced model $\mathcal{M}_N^m$ is induced as described in Definition 4.2.4. Therefore, from Lemma 4.2.5,

Lemma 4.5.2. *For any LTL property ϕ with atomic propositions that only refer to variables with indices less than m, for $m \geq 1$ if $N > m+1$ then $\mathcal{M}_N^m \models \phi$ implies $\mathcal{M}_N \models \phi$.*

In the following section we establish that the reduced model simulates the invariant model. However, notice that in the reduced model any local variable with an index greater than m does not correspond to a variable in the invariant model (and therefore the atomic propositions are not comparable). We resolve this issue by replacing the variables with index greater than or equal to m with a single variable (for example, by replacing $(\mathit{token}_m, \ldots, \mathit{token}_{N-1})$ by token_m).

Formally, for $m > 1$, $N > m$, let $\bar{\mathcal{M}}_N^m$ denote the model obtained from $\mathcal{M}_N^m$ by replacing every occurrence of

$$(\mathit{loc}_m, \mathit{loc}_{m+1}, \ldots, \mathit{loc}_{N-1}),$$
$$(\mathit{token}_m, \mathit{token}_{m+1}, \ldots, \mathit{token}_{N-1}),$$
$$\text{and } (\mathit{ch}[m], \mathit{ch}[m+1] \ldots, \mathit{ch}[N-1])$$

with the variables loc_m, token_m and *ch[m]* respectively. Then from Lemma 4.5.2 and variable renaming [49],

Lemma 4.5.3. *For any LTL property ϕ that refers only to the set of global variables, the local variables associated with the concrete processes and/or the channel variables, $\mathit{ch}[0], \mathit{ch}[1], \ldots, \mathit{ch}[m-1]$, then for $m \geq 1$ and $N > m+1$, if $\bar{\mathcal{M}}_N^m \models \phi$ then $\mathcal{M}_N \models \phi$.*

4.5.2 Simulation between the Reduced Model and the Invariant Model

We now show that a simulation relation exists between $\bar{\mathcal{M}}_N^m$ and $\mathcal{M}_I^m$. Let $\bar{\mathcal{M}}_N^m = (\bar{S}_N^m, \bar{s}_N^m, \bar{R}_N^m, \bar{L}_N^m)$ and $\mathcal{M}_I^m = (S_I^m, s_I^m, R_I^m, L_I^m)$ with atomic propositions $\bar{AP}_N^m$ and AP_I^m, respectively. Note that $\bar{AP}_N^m \supseteq \bar{AP}_I^m$. We define a relation H as follows,

$$\text{For all } s \in \bar{S}_N^m, s' \in S_I^m, H(s,s') \iff \bar{L}_N^m(s) \cap AP_I^m = L_I^m(s').$$

We show that $\bar{M}_N^m \preceq \mathcal{M}_I^m$ by establishing that H satisfies Definition 2.5.3. Immediately, condition 1 of Definition 2.5.3 holds by definition of the relation H. It remains to establish condition 2, i.e. to show that every transition in the reduced model is 'matched' by a transition in the invariant model. We establish the transitions that can be made in both the reduced and invariant models, by analysing the concrete and invariant specifications, respectively.

Note that it is only necessary to consider transitions in the concrete model that correspond to *atomic* sequences in the Promela code. Statements that are embedded in these are considered to be a single executable statement. Once an atomic sequence has been entered it will be executed completely unless one of the statements inside it blocks, in which case control can be passed to another process. In the CTRP example the only statements that can block are conditionals (i.e. propositions), channel read/writes or poll operations. Blocking statements are always put at the beginning of an atomic sequence so that it cannot be entered until the guard becomes true. Any atomic sequences that contain channel read/writes are also accompanied by an extra guard condition at the start of the atomic sequence to check that the appropriate channels are full/empty. Therefore, it is only ever necessary to consider the system state before an atomic sequence and after it.

Hence, by examining the atomic statements in the concrete Promela specification of the CTRP, the transitions that can be made in the underlying model, $\mathcal{M}_N$, can be determined. By considering the effect of the mapping h on the states of $\mathcal{M}_N$ we can therefore derive the transitions that can occur in $\bar{\mathcal{M}}_N^m$ and we can then check whether these transitions are matched by transitions in the invariant model. It is also necessary to establish the transitions that can be made in $\mathcal{M}_I^m$ by analysis of the atomic statements given in the invariant specification. Intuitively, each statement in the concrete specification of the *arbiter* process and of *process*$_i$, for $0 \leq i \leq m-1$ corresponds to the same statement in the invariant specification of the *arbiter* process and *process*$_i$ respectively. Also, each statement in the concrete specification of *process*$_i$, for $m \leq i \leq N-1$, corresponds to a statement in the invariant specification of *abstract_process*$_m$. We describe matching transitions in more detail below. Note that the atomic statements in the Promela code in Appendix A have been labelled with a number corresponding to each statement given below. We assume $\mathcal{M}_N = (S_N, s_0^N, R_N, L_N)$ with atomic propositions AP_N.

Statement 0. From the declaration of variables in the concrete specification we know that, in s_0^N, the initial state of $\mathcal{M}_N$, $L_N(s_0^N) = \{$*loc*$_i$*==idle*,*token*$_i$*==0*, *possess*$_i$*==false*, *working*$_i$*==false*, *end==false*, *done==false*, *work==false*, *ch*[i]=[] $|0 \leq i \leq N-1\}$. Note that for all $0 \leq i \leq N-1$, *loc*$_i$ is actually initially undefined, but we assume, for simplicity, that it has initial value *idle*. The process, *init*, declared in every specification, is the first process to be executed, executing atomic statement 0 and setting *ch[*0*]* to c for $c \geq 1$ and instantiating N *process*es, and one *arbiter* process. In the model $\mathcal{M}_N$ this will correspond to the transition $s_0^N \rightarrow t$ where $t \in S_N$ is labelled identically to s_0^N but with the proposition *ch[0]==[]* replaced by *ch[0]==[c]*.

Let $\bar{s}_0^m$ be the state in $\bar{\mathcal{M}}_N^m$ induced from the initial state of $\mathcal{M}_N$ under the surjective map h. Then we have, $\bar{L}_N^m(s_0^N) = \{$ *loc*$_i$*==idle*, *token*$_i$*==0*, *possess*$_j$*==false*, *working*$_j$*==false*, *end==false*, *done==false*, *work==false*, *ch[i]==[]* $|0 \leq i \leq m, 0 \leq j \leq m-1\}$. Similarly, let $\bar{t} = L_N^m(t)$ be the state induced from t, so the labelling of $\bar{t}$ will be identical to s' but with the proposition *ch[0]==[]* replaced by *ch[0]==[c]*. Note that $\bar{R}_N^m(s_0^m, \bar{t})$.

From the invariant specification, it is apparent that in the initial state s_0^I of the invariant model, $L_I^m(s_0^I) = \{$*loc*$_i$*==idle*, *token*$_i$*==0*, *possess*$_j$*==false*, *working*$_j$*==false*, *end==false*, *done==false*, *work==false*, *ch*[i]=[] $|0 \leq i \leq m, 0 \leq j \leq m-1\}$. Hence, $H(\bar{s}_0^m, s_0^I)$. The *init* process in the invariant specification runs m *process* processes, one *abstract_process* process and one *arbiter* process and sends c on *ch[0]*. This corresponds to the transition, $s_0^I \rightarrow t'$ in the model $\mathcal{M}_I^m$ where t' is labelled with the same propositions as s_0^I but with the proposition *ch[0]==[]* replaced by *ch[0]==[c]*. Clearly, $\bar{L}_N^m(\bar{t}) = L_I^m(t')$ and therefore $H(\bar{t}, t')$.

Statement 1. For all $0 \leq i \leq N-1$, atomic statement 1 in the concrete specification corresponds to *process*$_i$, with *loc*$_i$*==idle*, reading in a value, k say, for *token*$_i$ from its *in* channel (*ch[i]*), moving to *rcvd* and setting *possess*$_i$ to true. Note that this can only be executed if *ch[i]* is full. Thus, for any state s in $\mathcal{M}_N$, such that *loc*$_i$*==idle*, *ch[i]==k*$\in L_N(s)$, there exists a transition $s \rightarrow t$ such that *token*$_i$*==k*, *ch[i]==[]*, *loc*$_i$*==rcvd*, *possess*$_i$*==true*$\in L_N(t)$.

For $0 \leq i \leq m-1$, the surjective mappings (Section 4.5.1) on the variables mentioned are just the identity mappings. Thus, there exists a transition $\bar{s} \rightarrow \bar{t}$ in $\bar{M}_N^m$ where $\bar{s}$ and $\bar{t}$ are induced under h from s and t, respectively, such that *loc*$_i$*==idle*, *ch[i]==k*$\in L_N(s)$ and *token*$_i$*==k*, *ch[i]==[]*, *loc*$_i$*==rcvd*, *possess*$_i$*==true*$\in L_N(t)$.

Let s' be a state in the invariant model and suppose $H(\bar{s}, s')$. The labelling of $\bar{s}$ and s' is therefore identical and hence *loc*$_i$*==idle* and *ch[i]==k*$\in L_I^m(s')$. From statement 4 of the *process* specification in $\mathcal{I}(N, c)$, there must be a transition in $\mathcal{M}_I^m$ from s' to a state t' such that *token*$_i$*==k*, *ch[i]==[]*, *loc*$_i$*==rcvd*, *possess*$_i$*==true*$\in L_I^m(t')$ (and *loc*$_i$*==idle* and *ch[i]==k*$\notin L_I^m(t')$). Otherwise the labelling for t' is the same as for s'. Hence $L_I^m(t') = L_N^m(\bar{t})$ and so $H(\bar{t}, t')$.

Suppose now that $m \leq i \leq N-1$. Note that in any state v of $\mathcal{M}_N$, if *ch[i]==[k]*, *token*$_i$*==k* or *loc*$_i$*==rcvd*$\in L_N(v)$ then for all $j \neq i$, $m \leq j \leq N-1$, *ch[j]==[]*, *token*$_i$*==k*, *loc*$_i$*==rcvd*$\in L_N(v)$ respectively (since there is only one token). Let $\bar{s} \in \bar{S}_N^m$ be the state induced by s under h. Then *loc*$_m$*==idle*$\in L_N^m(s')$. The set of propositions *ch[m]==[]*, ..., *ch[*$i-1$*]==[]*, *ch[i]==[k]*, *ch[*$i+1$*]==[]*, ..., *ch[*$N-1$*]==[]*) belong to the labelling of s, and so (by the definition of h) the proposition *ch[m]==[k]*$\in L_N^m(s')$. If $\bar{t}$ is the state in $\bar{\mathcal{M}}_N^m$ induced by t under h we can show in a similar way that *token*$_m$*==k*, *ch[m]==[]*, *loc*$_m$*==rcvd*$\in \bar{L}_N^m(\bar{t})$. Note that a proposition corresponding to the tuple (*possess*$_m$, ..., *possess*$_{N-1}$) also belongs to $\bar{L}_N^m(\bar{t})$ but because every value of the tuple is mapped to *false*, the atomic proposition will be the same in every state of $\bar{M}_N^m$ and thus for the purposes of our analysis can be ignored. We have that $\bar{s} \rightarrow \bar{t}$.

Let s' be a state in the invariant model and suppose $H(\bar{s}, s')$. The labelling of $\bar{s}$ and s' is therefore identical with respect to AP_I^m and hence *loc*$_m$*==idle*, *ch[m]==k*$\in L_I^m(s')$. From statement 4 of the *abstract_process* specification in $\mathcal{I}(N, c)$, we have that *abstract_process*$_m$ reads a value from its *in* channel *ch[m]* and sets *possess*$_m$ to true. Thus, there must be a transition in $\mathcal{M}_I^m$ from s' to a state t' such that *token*$_m$*==k*, *ch[m]==[]*, *loc*$_m$*==rcvd* $\in L_I^m(t')$ (and *loc*$_m$*==idle* and *ch[i]==k*$\notin L_I^m(t')$). Otherwise the labelling for t' is the same as for s'. Hence $L_I^m(t') = L_N^m(\bar{t})$ and so $H(\bar{t}, t')$.

Statements 2 to 6. For each of the atomic statements 2 to 6 of $process_i$ $(0 \leq i \leq N-1)$ we can argue in a similar way as for statement 1 and show that, for a transition $s \rightarrow t$ in $\mathcal{M}_N$ that corresponds to a statement, the transition $s' \rightarrow t'$ in $\bar{M}_N^m$ induced by h is matched to a transition in $\mathcal{M}_I^m$ that corresponds to a statement of $process_i$ if $0 \leq i \leq m$, or to a statement of $abstract\_process_m$ if $m \leq i \leq N-1$.

Statement 7. Consider atomic statement 7 in the specification of the *arbiter* process. From the conditions of the statement, this can only be executed if *done* is *true* and *work* is *true*, in which case *work* is set to *false*. This corresponds to a transition $s \rightarrow t$ in $\mathcal{M}_N$, for any state $s \in S_N$ for which *work==true*$\in L_N(s)$ and *done==true*$\in L_N(s)$ and for a state $t \in S_N$ such that *work==false*$\in L_N(t)$, and otherwise the labelling is the same as for s.

Every value in the domains of *work* and *done* are mapped to the same values under the associated surjective mappings. Therefore, in the state, $\bar{s}$ induced by the mapping h, *work==true*$\in \bar{L}_N^m(\bar{s})$, and *done==true*$\in L_N^m(\bar{s})$. Similarly, for $\bar{t}$, the state induced by h, *work==false*$\in L_N^m(\bar{t})$ and otherwise the labelling is as for $\bar{s}$. By the definition of the transition relation of $\bar{M}_N^m$, there exists a transition $\bar{s} \rightarrow \bar{t}$.

Let s' be a state in the invariant model and suppose $H(\bar{s}, s')$. The labelling of $\bar{s}$ and s' is therefore identical and therefore *done==true* and *work==true*$\in L_I^m(s')$. From statement 5 of the *arbiter* process in the invariant specification, there must be a transition in $\mathcal{M}_I^m$ from s' to a state t' such that *work==false*$\in L_I^m(t')$ and, otherwise the labelling for t' is the same as for s'. Hence $L_I^m(t') = \bar{L}_N^m(\bar{t})$ and so $H(\bar{t}, t')$.

Statement 8. We can argue in a similar manner as for statement 7 to show that any transition in the reduced model induced by statement 8 (given in the specification of the *arbiter* process) is matched by a transition in $\mathcal{M}_I^m$ corresponding to statement 8 of *arbiter* in the invariant specification.

In Table 4.5.2 we give a summary of the transitions in $\mathcal{M}_N$ (in column two) that are derived from the atomic statements of *process* in the concrete specification (note that we do not consider transitions corresponding to the *arbiter* or *init* process in the table). The numbering in the leftmost column identifies each transition in the concrete model with a statement in *process* in the concrete specification. Each transition is described in the table by the value of the variables of a state for the transition to be enabled, a down arrow, and the value of some of the variables that belong to the state resulting from the transition. Note that we only consider the variables relevant to a transition.

In the table we also give the transitions in $\bar{\mathcal{M}}_N^m$ that correspond to transitions in the concrete model. We present these in the same manner as the transitions for $\mathcal{M}_N$. However, we split them into three columns depending on the value of i, where the concrete transition corresponds to the execution of a statement with respect to $process_i$. In the first column we describe the transition in $\bar{\mathcal{M}}_N^m$ if $process_i$ is a concrete process i.e. $0 \leq i \leq m-1$. In the second column we describe the transition in $\bar{\mathcal{M}}_N^m$ if $m \leq i < N-1$, and in the third column we describe the transition for $i = N-1$.

For the transitions in the reduced model, $\bar{\mathcal{M}}_N^m$, we do not include the values corresponding to the *possess* or *working* variables since these will be the same in every state of the reduced model. Excluding these values, the set of values are the same in $\bar{\mathcal{M}}_N^m$ and $\mathcal{M}_I^m$ (and therefore so are the sets of corresponding atomic propositions). Therefore, the transitions described for the reduced models correspond to the matching transitions in the invariant model.

Therefore, we have matched every transition in $\bar{M}_N^m$ to a transition in $\mathcal{M}_I^m$ i.e. we have established condition 2 of Definition 2.5.3. Therefore for all $N > m+1$, $\bar{\mathcal{M}}_N^m \preceq M_I^m$ and thus by Lemma 2.5.4 it follows that,

Lemma 4.5.4. *For $m > 1$, given a LTL property ϕ with atomic propositions in AP_I^m, $\mathcal{M}_I^m \models \phi \implies \bar{M}_N^m \models \phi$, for all $N > m+1$.*

Note that we can, in fact, prove a stronger result, showing that the relation H is a bisimulation relation. In general, when constructing an invariant in this manner, this is not the case.

Proof of Theorem 4.5.1

Proof. Let $m > 1$, $c \geq 1$, and let ϕ be an LTL property that does not contain any index greater than or equal to m in its atomic propositions. Suppose $\mathcal{M}_I^m \models \phi$. By Lemma 4.5.4, for all $N > m+1$, $\bar{M}_N^m \models \phi$. By Lemma 4.5.3, $\mathcal{M}_N \models \phi$ for all $N > m+1$. $\square$

Table 4.1: Transitions in $\mathcal{M}_N$, for process i, $0 \leq i \leq N-1$, and corresponding transitions in $\bar{\mathcal{M}}_N^m$, and, $\mathcal{M}_I^m$, $(c \geq t \geq 1)$.

	$\mathcal{M}_N$	$\bar{M}_N^m \setminus \mathcal{M}_I^m$ $0 \leq i \leq m-1$	$\bar{\mathcal{M}}_N^m \setminus \mathcal{M}_I^m$ $m \leq i < N-1$	$\bar{\mathcal{M}}_N^m \setminus \mathcal{M}_I^m$ $i = N-1$
1	$(idle, 0, F, F)$ $[t]_i$ $\downarrow$ $(rcvd, t, T, F)_i$ $[]_i$	$(idle, 0, F, F)_i$ $[t]_i$ $\downarrow$ $(rcvd, t, T, F)_i$ $[]_i$	$(idle, 0, F, F)_m$ $[t]_m$ $\downarrow$ $(rcvd, t, T, F)_m$ $[]_m$	$(idle, 0, F, F)_{N-1}$ $[t]_{N-1}$ $\downarrow$ $(rcvd, t, T, F)_{N-1}$ $[]_m$
2	$(rcvd, t, T, F)_i$ $!work, []_{i+1}$ $\downarrow$ $(idle, 0, F, F)_i$ $[t]_{i+1}$	$(rcvd, t, T, F)_i$ $!work, []_{i+1}$ $\downarrow$ $(idle, 0, F, F)_i$ $[t]_{i+1}$	$(rcvd, t, T, F)_m$ $!work, []_m$ $\downarrow$ $(idle, 0, F, F)_m$ $[t]_m$	$(rcvd, t, T, F)_{N-1}$ $!work, []_{N-1}$ $\downarrow$ $(idle, 0, F, F)_{N-1}$ $[t]_m$
3	$(rcvd, 1, T, F)_i$ $work, []_{i+1}$ $\downarrow$ $(finish, 0, T, T)_i$ $end, finished{+}1$	$(rcvd, 1, T, F)_i$ $work, []_{i+1}$ $\downarrow$ $(finish, 0, T, T)_i$ $end, finished + 1$	$(rcvd, 1, T, F)_m$ $work, []_m$ $\downarrow$ $(finish, 0, T, T)_m$ $end, finished + 1$	$(rcvd, 1, T, F)_{N-1}$ $work, []_{N-1}$ $\downarrow$ $(finish, 0, T, T)_{N-1}$ $end, finished{+}1$
4	$(rcvd, t, T, F)_i$ $work, []_{i+1}$ $\downarrow$ $(pass, t{-}1, T, T)_i$ $ready$	$(rcvd, t, T, F)_i$ $work, []_{i+1}$ $\downarrow$ $(pass, t{-}1, T, T)_i$ $ready$	$(rcvd, t, T, F)_m$ $work, []_m$ $\downarrow$ $(pass, t{-}1, T, T)_m$ $ready$	$(rcvd, t, T, F)_{N-1}$ $work, []$ $\downarrow$ $(pass, t{-}1, T, T)N-1$ $ready$
5	$(pass, t{-}1, T, T)_i$ $!work, ready, []_{i+1}$ $\downarrow$ $(idle, 0, F, F)_i$ $!ready, [t{-}1]_{i+1}$	$(pass, t{-}1, T, T)_i$ $!work, ready, []_m$ $\downarrow$ $(idle, 0, F, F)_m$ $!ready, [t{-}1]_m$	$(pass, t{-}1, T, T)_m$ $!work, ready, []_{i+1}$ $\downarrow$ $(idle, 0, F, F)_i$ $!ready, [t{-}1]_{i+1}$	$(pass, t{-}1, T, T)_m$ $!work, ready, []_m$ $\downarrow$ $(idle, 0, F, F)_m$ $!ready, [t{-}1]_m$
6	$(finish, 0, T, T)_i$ $\downarrow$ $(finish, 0, T, T)_i$	$(finish, 0, T, T)_m$ $\downarrow$ $(finish, 0, T, T)_m$	$(finish, 0, T, T)_i$ $\downarrow$ $(finish, 0, T, T)_i$	$(finish, 0, T, T)_m$ $\downarrow$ $(finish, 0, T, T)_m$

Table 4.2: Description of variables corresponding to entries in Table 4.5.2

Decription	Variables	Example table entries
Local state of process i	$(loc_i, token_i, possess_i, working_i)$	$(idle, 0, F, F)_i$
Channel i	$ch[i]$	$[t]_i$, $[]_i$
Global boolean variables	$end, work, ready$	$end, !work$
Global integer variables	$finished$	$finished{+}1$

4.6 A Probabilistic Model of the CTRP

In Section 4.3 we introduced the CTRP, a token ring protocol that limits the number of operations performed by processes using a counter. We modelled the CTRP using a central arbiter process that non-deterministically determines whether a process can perform an operation or not when it receives the token. This could, for example, correspond to intermittent process failure or to whether the 'queue' of operations at a process is empty or not when it receives the token. We now consider an example where we have some additional knowledge about the protocol, namely the *probability* of whether a process can perform an operation or not. For example, we might know the probability of a processes queue being empty or the probability of a process having failed, when it receives the token. Therefore, we model the protocol probabilistically, allowing each process to make a probabilistic choice whenever it receives the token.

Note that in the non-probabilistic model of the CTRP there exists an execution path such that the token can be passed around the ring forever without being decremented. However, if the choice is made probabilistically, the probability that the token is never decremented is equal to zero, and therefore we do not need to apply fairness conditions.

We assume that the scheduling of the processes is still non-deterministic and therefore we model the protocol as a MDP, which we specify using PRISM. An example of a PRISM specification of the probabilistic CTRP is given in Appendix C and described in Section 4.8.

4.7 Data Abstraction for MDPs by Partitioning

We now define data abstraction for MDPs, starting with a definition of a MDP over a variable set.

Definition 4.7.1. *Let X be a set of variables with domain $D(X)$ where the tuple of initial values for the variables is given by $init(X)$. A (labelled) MDP $\mathcal{M}$ over X is a tuple $(D(X), init(X), Steps, L)$, where L labels each state with atomic propositions from the set AP over X.*

We give a definition of a reduced MDP under 'data abstraction', based on the definition of a *quotient* MDP of D'Argenio et al. [24]. A quotient MDP partitions the state space.

Definition 4.7.2. *[24] Let $\mathcal{M} = (S, s_0, Steps, L)$ be a MDP and let $\mathcal{Q} = (A_k)_{k \in K}$ be a partition of S. The quotient MDP according to $\mathcal{Q}$ is the MDP $\mathcal{M} \setminus \mathcal{Q} = (\mathcal{Q}, Q_0, Steps', L')$ where*

1. *$Q_0 \in \mathcal{Q}$ is such that $s_0 \in Q_0$,*
2. *For $Q \in \mathcal{Q}$, $\mu' \in Steps'(Q) \Leftrightarrow \exists s \in Q : \mu \in Steps(s) \wedge \forall Q' \in \mathcal{Q}$, $\mu'(Q') = \sum_{s' \in Q'} \mu(s')$,*
3. *For all $Q \in \mathcal{Q}$, $L'(Q) = \cup_{s \in Q} L(s)$.*

A reduced MDP under data abstraction is a quotient MDP, where the partitioning is given according to the states for which the variables have the same abstract values. Note that each state of a MDP over a variable set X is a tuple from $D(X)$ and therefore we can apply a mapping on $D(X)$ directly to the state.

Definition 4.7.3. *Let $\mathcal{M} = (S, s_0, Steps, L)$ be a MDP over set of variables X. Let AP be the set of atomic propositions over X. If $D'(X)$ is an abstract domain of X and h a surjection from D to D' then let H be the equivalence relation defined by $(s, t) \in H$ if and only if $h(s) = h(t)$. Let $\mathcal{Q}$ denote the partition defined by H. Then the reduced MDP under h is the quotient MDP $\mathcal{M} \setminus \mathcal{Q}$.*

For any two partitions $\mathcal{Q}$ and $\mathcal{T}$ of the same set, let $\mathcal{Q} \leq \mathcal{T} \Leftrightarrow \forall Q \in \mathcal{Q}, \exists T \in \mathcal{T}, Q \subseteq T$. Furthermore, for a MDP $\mathcal{M} = (S, s_0, Steps, L)$ with atomic propositions AP, a state $s \in S$ and $\phi \in 2^{AP}$ let,

$$p_s^{inf}(\phi) \triangleq \inf_{A \in Adv_{\mathcal{M}}} \{Prob_s^A(\{\omega \in Path^A(s) | \omega \models \Diamond\phi\})\} \text{ and,}$$

$$p_s^{sup}(\phi) \triangleq \sup_{A \in Adv_{\mathcal{M}}} \{Prob_s^A(\{\omega \in Path^A(s) | \omega \models \Diamond\phi\})\}.$$

Lemma 4.7.4 is a restriction of a result given in [24]. Specifically, we do not define an initial condition on states, as is the case in [24], but consider just the initial state.

Lemma 4.7.4. *Let $\mathcal{M} = (S, s_0, Steps, L)$ be a MDP with set of atomic propositions AP. Let $\phi \in 2^{AP}$ and let C_ϕ be the equivalence relation on S defined by:*

$$(s, t) \in C_\phi \Leftrightarrow ((s \not\models \phi \wedge s = s_0 \Leftrightarrow t \not\models \phi \wedge t = s_0) \wedge (s \models \phi \Leftrightarrow t \models \phi)).$$

Let $\mathcal{C}$ denote the partition defined by C_ϕ, then for any two partitions of S, $\mathcal{Q}$ and $\mathcal{T}$, such that $\mathcal{Q} \leq \mathcal{T} \leq \mathcal{C}$,

$$p_{Q_0}^{sup}(\phi) \leq p_{T_0}^{sup}(\phi) \text{ and } p_{Q_0}^{inf}(\phi) \geq p_{T_0}^{inf}(\phi),$$

where Q_0, T_0 are the initial states of $\mathcal{M} \setminus \mathcal{Q}$ and $\mathcal{M} \setminus \mathcal{T}$ respectively.

Note that we can partition a set of states into singleton sets, in which case the quotient MDP is isomorphic to the original MDP [24]. We will use this result in the proof of Lemma 4.7.5.

Lemma 4.7.5. *Let $\mathcal{M} = (S, s_0, Steps, L)$ be a MDP over variable set X with set of atomic propositions AP over X. Let h be a surjective mapping on $D(X)$ such that for all $s \in S$, $s \neq s_0$, $h(s) \neq h(s_0)$. Let $\mathcal{M}^r$ denote the reduced MDP under h. Let ϕ be a PCTL-reachability property that only contains propositions of the form $(x_i = v)$ where x_i a variable in X, v belongs to the abstract domain of x and $h_i^{-1}(v) = \{v\}$ (for h_i, the surjection on the domain of x_i) then $\mathcal{M}^r \models \phi \Rightarrow \mathcal{M} \models \phi$.*

Proof. Let $\mathcal{T}$ denote the partition that partitions S into singleton sets and let H denote the equivalence relation defined by h as $H(s,t) \Leftrightarrow h(s) = h(t)$. Let $\mathcal{Q}$ be the partition defined by H. The property ϕ has the form $\mathcal{P}_{\bowtie=p}[\Diamond\psi]$. By definition of PCTL-reachability, $\psi \in 2^{AP}$. Let C_ψ denote the equivalence relation defined as for Lemma 4.7.4 and $\mathcal{C}$ the partition defined by C_ψ. Clearly $\mathcal{T} \leq \mathcal{Q}$.

We establish that $\mathcal{Q} \leq \mathcal{C}$. It should be apparent that either $\mathcal{C} = \{C_0, C_n, C_y\}$ if $s_0 \not\models \psi$ or $\mathcal{C} = \{C_n, C_y \cup C_0\}$ if $s_0 \models \psi$ where $C_0 = \{s_0\}$, $C_n = \{s \in S | s \neq s_0 \wedge s \not\models \psi\}$, $C_y = \{s \in S | s \neq s_0 \wedge s \models \psi\}$.

Let $Q \in \mathcal{Q}$. Suppose $Q = Q_0$, the initial state of $\mathcal{M}^r$. Since for all $s \in S$, $s \neq s_0$, $h(s) \neq h(s_0)$ then $Q_0 = \{s_0\}$. If $C_0 \in \mathcal{C}$ then $Q_0 \subseteq C_0$, otherwise $Q_0 \subseteq C_y \cup C_0$.

Suppose $Q \neq Q_0$ and let $s \in Q$. If $s \models \psi$ then for all $t \in Q$, $t \models \psi$ since $h(s) = h(t)$ and ϕ only contains propositions of the form $(x_i = v)$ where $h_i^{-1}(v) = \{v\}$. Thus $Q \subseteq C_y$. Similarly if $s \not\models \psi$ then for all $t \in Q$, $t \not\models \psi$ and hence $Q \subseteq C_n$.

Thus $\mathcal{T} \leq \mathcal{Q} \leq \mathcal{C}$ and by Lemma 4.7.4 it follows that if $Q_0 = \{s_0\} \models \phi$ then $T_0 = \{s_0\} \models \phi$. Since $\mathcal{M} \setminus \mathcal{T}$ is isomorphic to $\mathcal{M}$, we have that if $\mathcal{M}^r \models \phi$ then $\mathcal{M} \models \phi$. $\square$

4.8 Specification of a Randomised CTRP

We specify concrete probabilistic models of the CTRP in PRISM that are similar to the models of the CTRP specified in Promela. Each PRISM specification consists of N 'process' modules, *process1*, *process2*, …, *processN* but because we model a process' behaviour by probabilistic choice we do not require an *arbiter* process. An example of a PRISM specification for the protocol for $N = 4$ and $c = 5$ is shown in Appendix C. Note that for a process, *processi*, we use the notation $process_i$. Similarly for any variable, *vari*, we use the notation var_i.

Consider first module $process_0$. The local variables declared in the body of this module are loc_0, $token_0$, $possess_0$ and $working_0$, which are similar to the those described in the Promela specifications. Note that we have to declare loc_0 as a variable since PRISM uses a guarded command style language and thus does not include program counters. We declare constants *idle*, *rcvd*, *pass*, *finish* and declare loc_0 to have domain in the range of these constants. The variables $possess_0$ and $working_0$ are booleans.

The global variables are *end* and *finished.* Although PRISM does not provide explicit support for the specification of channels, because we only consider channels of length one, we can declare a set of global variables to represent the channels. Specifically, we define the variables ch_0, ch_1, …, ch_{N-1}. Each of these has domain {0,1,…,c}. Notice that once the counter equals zero the game ends and therefore a channel will never take the value 0. We can therefore use 0 to represent the empty channel.

Although specified differently, the behaviour of $process_0$ is similar to that of $process_0$ from the Promela specification. For $process_0$, initially loc_0=*idle.* In this state, if ch_0*!=0*, then $process_0$ can store the value of ch_0 in $token_0$, setting ch_0 to be 0 (i.e. empty) and setting the variable $possess_0$ to be true and loc_0 to be *rcvd.*

The main difference between the behaviour of the Promela specification and the PRISM specification is when loc_0=*rcvd.* From this state the (PRISM) process will make a probabilistic choice, decrementing the counter with probability $\frac{1}{2}$ or simply passing the counter to $process_1$ (setting ch_0=$token_0$) with probability $\frac{1}{2}$. If $process_0$ chooses to decrement the counter it will set loc_0=*pass* if the result of decrementing $token_0$ is 0. Otherwise it will set loc_0=*finish*, in which state the process will loop. From the state loc_0=*pass*, the process will then set ch_0=$token_0$ and loc_0=$idle$.

Note that the choice of the probabilities is arbitrary. However, (in order for us to be able to construct an invariant in the manner described in the subsequent section) the probabilities must be identical and must not depend on the number of processes.

Modules $process_1$, $process_2$, ..., $process_{N-1}$ can be constructed by renaming the variables of $process_0$. We let $\mathcal{S}(N, c)$ denote the PRISM specification with ch_0 initialised to c and $N - 2$ renamed 'process' modules.

4.9 Constructing an Invariant for a Randomised Model of the CTRP

The PRISM specification of an invariant model for the CTRP with two concrete processes $process_0$, $process_1$ and an invariant process $abstract\_process_2$ is given in Appendix D. The invariant process is constructed in a similar manner to that of the invariant process described for the Promela specification of the CTRP, except that we need to take into account the probabilistic choices made by the concrete processes. We let $\mathcal{I}(N, c)$ denote the PRISM specification with ch_0 initialised to c, $m - 1$ renamed 'process' modules and one $abstract\_process_m$ module.

4.9.1 Properties of the Randomised CTRP

All of the properties given below are specified in the PCTL temporal logic defined in Section 3.3 and have been verified for models of the protocol for rings of size two to five, with a counter with initial value of one to ten. These properties have also been verified for the invariant specification of the CTRP using PRISM. This has been done for a counter with initial value between one to ten and with two concrete processes. Note that these properties are PCTL-reachability properties.

M1. With probability 1 some process will finish.

$$\mathcal{P}_{\geq 1}[true\mathcal{U}(end)]$$

M2. With probability at most 0.7, process 1 will finish.

$$\mathcal{P}_{\leq 0.7}[\mathit{true}\,\mathcal{U}(\mathit{loc}_1 = \mathit{finish})]$$

M3. With probability at least 0.5 eventually process 1 will receive the token.

$$\mathcal{P}_{\geq 0.5}[\mathit{true}\,\mathcal{U}(\mathit{possess}_1)]$$

4.10 Parameterised Verification of the Randomised Model of the CTRP

We give a theorem (and proof) similar to Theorem 4.5.1. For $N > 1$ and $c > 1$, given a PRISM specification, $\mathcal{S}(N, c)$, of the CTRP, the *concrete* model of the CTRP, with $N > 1$ *process* instantiations, is given by $\mathcal{M}_N = \mathcal{M}(\mathcal{S}(N, c))$ where $\mathcal{M}_N$ is derived from $\mathcal{S}(N, c)$ according to the semantics of PRISM with respect to MDPs [1]. Given a PRISM invariant specification $\mathcal{I}(m, c)$, for $m > 1$, let $\mathcal{M}_I^m = \mathcal{M}(\mathcal{I}(m, c))$ be defined as the *invariant* model of the CTRP.

Theorem 4.10.1. *Let $m > 1$ and let ϕ be a PCTL-reachability property that does not contain any index greater than or equal to m in its atomic propositions. Then, for all $N > m$, $c \geq 1$, if $\mathcal{M}_I^m \models \phi$ then $\mathcal{M}_N \models \phi$.*

4.10.1 Creating Reduced Models by Data Abstraction

Let $m > 1$ and $N > m$. As for the proof of Theorem 4.5.1, in order to show Theorem 4.10.1 we define a reduced structure based on the concrete model constructed by data abstraction. This is done by defining a set of surjections that map the values from the domain of variables of every concrete model, $\mathcal{M}_N$, to values in an abstract domain. From the specification $\mathcal{S}(N, c)$, we know that the set of variables in the concrete model $\mathcal{M}_N$ is identical to the set of variables defined for the Promela specification of the CTRP. These surjections are defined to be identical to those in Section 4.5.1. The reduced model $\mathcal{M}_N^m$ is then induced as described in Definition 4.7.3, and from Lemma 4.7.5,

Lemma 4.10.2. *For any PCTL-reachability property ϕ, for $m \geq 1$ and $N > m + 1$, if $\mathcal{M}_N^m \models \phi$ then $\mathcal{M}_N \models \phi$.*

For $m > 1$, $N > m$, let $\bar{M}_N^m$ denote the model obtained from $\mathcal{M}_N^m$ by substituting variables in an identical manner as in Section 4.5.1. Then by Lemma 4.10.2,

Lemma 4.10.3. *For any PCTL-reachability property ϕ that refers only to the set of global variables, the local variables associated with the concrete processes and/or the channel variables $ch[0], ch[1], \dots, ch[m-1]$, for $m \geq 1$ and $N > m+1$, if $\bar{M}_N^m \models \phi$ then $\mathcal{M}_N \models \phi$.*

4.10.2 Bisimulation between the Reduced Models and the Invariant Model

In Section 4.5.2 we established a simulation relation between the (non-probabilistic) reduced models and the (non-probabilistic) invariant model for the CTRP. In this section we prove a stronger result showing that there exists a bisimulation between the probabilistic invariant model and the probabilistic reduced models.

Let $\bar{M}_N^m = (\bar{S}_N^m, \bar{s}_N^m, \bar{Steps}_N^m, \bar{L}_N^m)$ and $\mathcal{M}_I^m = (S_I^m, s_I^m, Steps_I^m, L_I^m)$ have atomic propositions $\bar{AP}_N^m$ and AP_I^m, respectively. Note that $\bar{AP}_N^m \supseteq \bar{AP}_I^m$. Let $\mathcal{M}_N^* = (S^*, S_0^*, Steps^*, L^*)$ be the combination of $\mathcal{M}_N^m$ and $\mathcal{M}_I^m$ We can define a relation H as follows,

$$\text{For all } s, s' \in S^*, H(s, s') \iff L^*(s) = L^*(s')$$

We show that $\bar{M}_N^m \approx \mathcal{M}_I^m$ by establishing that H satisfies Definition 3.5.1 for $\mathcal{M}_N^*$. Immediately, condition 1 of Definition 3.5.1 holds by definition of the relation H. It remains to establish condition 2, i.e. to show that every transition in the reduced model is 'equivalent' to a transition in the invariant model. We establish the transitions that can be made in both the reduced and invariant models, by analysing the concrete and invariant specifications, respectively. We assume $\mathcal{M}_N = (S_N, s_0^N, R_N, L_N)$ has atomic propositions AP_N.

Note that the majority of transitions in the concrete, invariant and reduced MDP models are non-probabilistic i.e. occur with probability 1. To illustrate how we can establish that H is a bisimulation we consider just one of the probabilistic transitions. Note that the statements of the PRISM concrete specifications and invariant specifications are numbered in Appendices C and D respectively.

Statement 3. For all $0 \leq i \leq N-1$, statement 3. in the concrete specification corresponds to $process_i$, with loc_i=*rcvd*, choosing with equal probability whether to decrement the token counter, moving to *pass* and setting *working$_i$* to true or to pass on the counter directly, moving to a state with *loc$_i$ =idle*. Note that this statement can only be executed if *ch*$_{i+1}$ is empty and the value of *token$_i$* is greater than one. Thus, for any state s in $\mathcal{M}_N$, such that *loc$_i$=rcvd*, *ch$_i$*=0, *token$_i$=k* $\in L_N(s)$ (for $k > 1$), there exists transitions $s \xrightarrow{\mu} t_1$ and $s \xrightarrow{\mu} t_2$ such that for $t_1, t_2 \in S$, $\mu(t_1) = \mu(t_2) = 0.5$ with *token$_i$=k-1*, *ch$_i$=0*, *loc$_i$=pass*, *working$_i$=true*$\in L_N(t_1)$ and *ch$_i$=k*, *loc$_i$=idle*, *possess$_i$=false* and *token$_i$=0*$\in L_N(t_2)$.

For $0 \leq i \leq m-1$, the surjective mappings defined on the variables mentioned are just the identity mappings. Thus, there exists transitions $\bar{s} \xrightarrow{\mu'} \bar{t_1}$ and $\bar{s} \xrightarrow{\mu'} \bar{t_2}$ in $\bar{M}_N^m$ where $\bar{s}$, $\bar{t_1}$ and $\bar{t_2}$ are induced under h from s, t_1 and t_2 respectively such that $\mu'(\bar{t_1}) = \mu'(\bar{t_2}) = 0.5$. It should be clear that *loc$_i$=rcvd*, *ch$_i$*=0, *token$_i$=k* $\in L_N^m(\bar{s})$ and *token$_i$=k-1*, *ch$_i$=0*, *loc$_i$=pass*, *working$_i$=true*$\in$ $L_N^m(\bar{t_1})$ and *ch$_i$=k*, *loc$_i$=idle*, *possess$_i$=false* and *token$_i$=0*$\in L_N^m(\bar{t_2})$.

Let s' be a state in the invariant model and suppose $H(\bar{s}, s')$. The labelling of $\bar{s}$ and s' with respect to AP_I^m is therefore identical and hence *loc$_i$=rcvd*, *token$_i$=k* and *ch$_i$*=0$\in$ $L_I^m(s')$. From statement 3. of the *process* specification in $\mathcal{I}(N, c)$, there must be a transition in $\mathcal{M}_I^m$ from s' to states t'_1 and t'_2 under distribution μ'' such that $\mu''(t'_1) = \mu(t'_2) = 0.5$. We have that *token$_i$=k-1*, *ch$_i$=0*, *loc$_i$=pass*, *working$_i$=true*$\in L_I^m(t'_1)$ and *ch$_i$=k*, *loc$_i$=idle*, *possess$_i$=false* and *token$_i$=0*$\in L_I^m(t'_2)$. Otherwise the labelling for t'_1 and t'_2 is the same as for s'. Hence $L_I^m(t'_1) = L_N^m(\bar{t_1}) \cap AP_I^m$ and $L_I^m(t'_2) = L_N^m(\bar{t_2}) \cap AP_I^m$ and so $H(\bar{t_1}, t'_1)$ and $H(\bar{t_2}, t'_2)$. Let $[s]_H$ denote the equivalence class of state s under H then $\mu'([t'_1]_H) = \mu''([t'_1]_H = 0.5$ and $\mu'([t'_2]_H) = \mu''([t'_2]_H) = 0.5$.

Suppose now that $m \leq i \leq N-1$. Note that in any state v of $\mathcal{M}_N$, if *loc$_i$=rcvd*, *ch$_i$*=0, *token$_i$=k* $\in L_N(v)$ then for all $j \neq i$, $m \leq j \leq N-1$, *ch$_j$=0*, *token$_j$*=0, *loc$_j$=idle*$\in L_N(v)$ respectively (since there is only one token). Let $\bar{s} \in \bar{S}_N^m$ be the state induced by s under h. The set of propositions (*loc$_m$=idle*, ..., *loc$_{i-1}$=idle*, *loc$_i$=rcvd*, *loc$_{i+1}$=idle*, ..., *loc$_{N-1}$=idle*) belong to the labelling of s, and so (by the definition of h) the proposition *loc$_m$=rcvd*$\in$ $L_N^m(s')$. Similarly *token$_m$=k*$\in$ $L_N^m(s')$. If $\bar{t_1}$ and $\bar{t_2}$ are the states in $\bar{\mathcal{M}}_N^m$ induced by t_1 and t_2 under h respectively we can show in a similar way that *token$_m$*$=k-1$, *ch$_m$=0*, *loc$_m$=pass*$\in$ $\bar{L}_N^m(\bar{t_1})$ and *ch$_m$=k*, *loc$_m$=idle* and *token$_m$=0*$\in$ $L_N^m(t'_2)$. Note that a proposition corresponding to the tuple (*possess$_m$*, ..., *possess$_{N-1}$*) also belongs to $\bar{L}_N^m(\bar{t_1})$ and $\bar{L}_N^m(\bar{t_1})$ but because every value of the tuple is mapped to *false*, the atomic proposition will be the same in every state of $\bar{M}_N^m$ and thus for the purposes of our analysis can be ignored. We have that $\bar{s} \xrightarrow{\mu'} \bar{t_1}$ and $\bar{s} \xrightarrow{\mu'} \bar{t_2}$ where $\mu'(\bar{t_1}) = \mu'(\bar{t_2}) = 0.5$.

Let s' be a state in the invariant model and suppose $H(\bar{s}, s')$. The labelling of $\bar{s}$ and s' is therefore identical with respect to AP_I^m and hence *loc*$_m$*=rcvd*, *ch[m]==0*$\in L_I^m(s')$.

Suppose that $m < N - 1$. From statement 3.1 of the *abstract_process* specification in $\mathcal{I}(N, c)$, *abstract_process*$_m$ can probabilistically choose to send a value on channel *ch*$_m$ or decrement the counter. Thus, there must be transitions in $\mathcal{M}_I^m$ from s_1' to states t_1' and t_2' such that *token*$_m$*=k-1*, *ch*$_m$*=0*, *loc*$_m$*=pass* $\in L_I^m(t_1')$ and *loc*$_m$*=idle* and *ch*$_m$*=k*$\in L_I^m(t_2')$). Otherwise the labelling for t_1' and t_2' are the same as for s'. Hence $L_I^m(t_1') = L_N^m(\bar{t})$ and $L_I^m(t_2') = L_N^m(\bar{t})$ and so $H(\bar{t}, t_1')$ and $H(\bar{t}, t_2')$. Thus $\mu'([t_1']_H) = \mu''([t_1']_H = 0.5$ and $\mu'([t_2']_H) = \mu''([t_2']_H = 0.5$.

Similarly, if $m = N - 1$ we can show that, from statement 3.2 of the definition of *abstract_process*$_m$, there exists transitions that are equivalent to those in the reduced model.

By considering each of the transitions in the reduced and invariant models and arguing in a similar manner to the above we can establish that condition 2 of Definition 3.5.1 holds for H and therefore that $\bar{\mathcal{M}}_N^m$ and $\mathcal{M}_I^m$ are bisimilar. Notice that $\bar{s_0} \approx s_0^I$ and hence, from Lemma 3.5.2 it follows that,

Lemma 4.10.4. *For $m > 1$, given a PCTL property ϕ with atomic propositions in AP_I^m, $\mathcal{M}_I^m \models \phi \implies \bar{M}_N^m \models \phi$, for all $N > m + 1$.*

Proof of Theorem 4.10.1

Proof. Let $m > 1$, $c \geq 1$, and let ϕ be a PCTL-reachability property that does not contain any index greater than or equal to m in its atomic propositions. Suppose $\mathcal{M}_I^m \models \phi$. By Lemma 4.10.4, for all $N > m + 1$, $\bar{M}_N^m \models \phi$. By Lemma 4.10.3, therefore $\mathcal{M}_N \models \phi$ for all $N > m + 1$. □

4.11 Discussion

Proving properties of the CTRP for arbitrary counter values The focus of our work on parameterised verification is on the analysis of distributed systems parameterised by the number of processes. However, sometimes it is necessary to consider other data values that are parameterised. For example, in the CTRP the initial value of the token is a parameter for the models. In particular, the counter is a parameter in the invariant specification, thus we have established a parameterised family of invariants over a 'lattice' of models. If we fix a value for m (the number of concrete processes), then we can consider the parameterised family of models $\mathbf{M} = \{\mathcal{M}(c) = \mathcal{M}(\mathcal{I}(m, c)) | c > 1\}$. We could use one of two approaches to parameterised verification of this family. One, by data abstraction, we define a surjective mapping for every token variable and every channel variable such that any value greater than two is mapped to the value two and thus provides a simulation relation. As we will discuss in further detail, below, this introduces additional non-determinism to the model. In particular, the counter value can now remain at two forever, even in the presence of weak fairness, so we would need to introduce some extra fairness constraints. Two, by induction on the models: we show that $\mathcal{M}(c)$ is equivalent to $\mathcal{M}(c+1)$, ignoring the variables with counter value greater than 1. This approach is explored in Chapters 5, 6 and 7.

Verification of Liveness Properties One of the limitations of the construction of an invariant via the data abstraction approach is that it is not guaranteed to preserve liveness properties. Intuitively, this is due to the additional non-determinism introduced by the abstraction process. For example, the application of abstraction to the CTRP leads to a cycle in the underlying model in which the *abstract_process* passes the counter to itself an infinite number of times. This can lead to certain liveness properties failing to hold. The introduction of this cycle is necessary to represent the behaviour of an arbitrary number of *process*es passing the counter to each other, but it should be apparent that it should only represent a *finite* number of processes.

We should therefore only need to consider paths in the invariant model in which the abstract process will eventually pass the token to $process_0$. That is, we should add fairness constraints to the model. However, because the non-deterministic choice is made within the *abstract_process*, the weak fairness option of SPIN is not strong enough to guarantee that the token will eventually be passed to $process_0$. (Note, though, that it does guarantee that the *arbiter* process will eventually schedule an operation and, therefore, the counter will be guaranteed to eventually equal zero even if it is always retained by the *abstract_process*). Hence, we have to derive additional fairness constraints for the models, using LTL. For example, we can express the requirement that $abstract\_process_m$ eventually passes the counter to $process_0$ as the LTL path formula, $\varphi \equiv \Box(ch[m] > 0 \implies \Diamond(ch[0] > 0 || end))$, and then, when verifying a property ϕ, prefix the property with the above requirement as follows, $\varphi \implies \phi$. In this manner we can establish properties of the invariant model of the CTRP that otherwise would not hold. For example, we can show that the invariant model $\mathcal{M}_I^2$ with $c < 10$ satisfies $\varphi \implies \phi$.

However, simulation does not guarantee preservation of properties under the assumption of fairness. Therefore, we cannot draw conclusions about whether or not properties are satisfied in the concrete or reduced models under fairness by showing that they are satisfied in the invariant model under fairness. A solution to this is to show that there exists a *fair simulation* between the invariant model under fairness and the reduced model, under the same fairness constraints. The difficulty in establishing this is that, since fairness relates to infinite executions, we cannot just consider transitions, we must consider paths. Specifically, we must show that for every fair path of a model there exists a fair path in the model that simulates it.

Note that we would also have to prove that data abstraction preserves fairness, i.e. that if a reduced model satisfies a property under fairness then the original model also satisfies the property under fairness. Work has already been carried out on this problem. For example, in [20], an abstraction method based on the μ-calculus temporal logic is described for which it has been proved that liveness certain liveness properties still hold under abstraction.

In the case of the randomised model of the CTRP note that, for the set of paths along which the abstract process only passes the counter to itself, we reach a state where the counter is zero and the protocol finishes with probability one. Therefore, the probability of the abstract process passing the counter to itself forever equals zero. However, in other examples we might still need to employ fairness in a similar manner to that described above. Fairness for MDPs is defined, for example, in [10]. To our knowledge, fair simulation for MDPs has not been defined.

Game-based Abstraction In Section 4.7 we considered data abstraction for MDPs based on the partitioning of a MDP [24]. It may be possible to extend this approach, using the game-based abstraction approach of Kwiatkowska et al. [41]. They consider a partition of the state space but represent the quotient MDP by a two-player stochastic game, where one player resolves the non-determinism of the concrete model and the other player resolves the non-determinism introduced by the abstraction. By considering the players in co-operation, a lower bound for the minimum probability of some reachability property being satisfied and an upper bound for the maximum probability of a reachability property being satisfied can be determined. Note that these could equally be achieved by considering the quotient MDP directly. However, they also derive an upper bound for the minimum probability and a lower bound for the maximum probability by considering the players in competition. This approach therefore gives a better approximation of the probability of some property holding in a concrete model. A prototype model checking tool has been implemented for this type of analysis. By using this tool to distinguish between the non-determinism in the concrete models and the non-determinism introduced by the data abstraction step, it would be possible to establish bounds on the minimum and maximum probabilities of some property holding of an invariant. This would provide more information on the probability of properties holding in the concrete models.

An SMV Specification of the Parameterised Token Ring Protocol We have also created a set of SMV specifications of the token ring system. The SMV versions are as close to the Promela coding as possible, but the differences between the modelling languages means that they are not identical. For example, it is not possible to define channels in SMV. Nonetheless, the behaviour of the SMV models are nearly identical to that of the Promela models. If an invariant specification is constructed that is similar to the Promela invariant specification, the proof described above can be modified in a simple manner to prove a similar result for the SMV concrete and invariant specifications. Thus, the methodology described above is not restricted to a particular modelling language. This means that we can employ features of different model-checking tools for proving properties of the invariant specification. For example, SMV allows CTL properties to be checked and a variety of fairness constraints to be specified.

Related Work A solution to the parameterised model checking problem for token ring protocols is presented in [29]. By establishing an equivalence relation between rings of different size, it is shown that, for certain forms of indexed CTL*\X properties, if a property holds for a ring of size equal to some cutoff value then it will hold for any size of ring. Model checking can be used to verify the property for a ring of a given size and it can then be deduced that the property holds for an arbitrary sized ring. Token ring protocols are only considered in which the counter consists of a single value. In fact, it is proven that the parameterised model checking problem for token rings where the token can have more than one value is undecidable. Therefore, the approach they describe is not applicable to the CTRP described here.

In a further approach to parameterised verification based on abstraction [53], a counter abstraction is used: counting the number of processes in each local state and limiting the counter values to a maximum value of two. The method is used for proving properties of non-probabilistic systems and for proving qualitative properties of probabilistic systems. A method is also described for deriving fairness requirements. Proving properties of the abstracted model under a set of derived 'abstract' fairness constraints ensures that the property holds in the concrete model, thus enabling verification of liveness properties. The approach has the advantage of being fully automated but is applicable only to fully symmetric systems and therefore would not be applicable in the case of systems with a ring topology as for the CTRP.

Summary In this chapter we have constructed network invariants for a parameterised family of models, through the use of data abstraction. We have applied this method to a simple counter token ring protocol, specifying a parameterised family of models for the protocol and an invariant model for the family using Promela. We have established that any LTL property that is satisfied by the invariant is satisfied by any model in the parameterised family of models. This result was then extended for a parameterised family of probabilistic models of the protocol using previous results on abstraction for probabilistic models. The family of models and the invariant model in this case were specified using PRISM.

Chapter 5

Probabilistic Parameterised Verification of Deterministically Degenerative Systems

Outline We present an inductive proof schema for establishing properties of randomised distributed systems that are deterministically degenerative. We define a deterministically degenerative family of MDP models parameterised by a communication topology and show that, for a certain class of QLTL properties, if a property is satisfied by a 'base' subset of the family of models then it is true for every model in the family. We demonstrate our approach by considering the IEEE 1394 Firewire Tree Identify Protocol.

5.1 Introduction

In this chapter and Chapter 6 we provide a technique for reasoning about the parameterised model checking problem that is sound and complete for a class of parameterised probabilistic system. In particular, we tackle the PMCP for randomised distributed systems by extending an inductive proof that was introduced for a non-probabilistic parameterised distributed system (the IEEE 1394 Firewire tree identify protocol) [49]. In [49] it is demonstrated that any configuration of the Firewire system will eventually behave like a smaller system, describing this behaviour as *degenerative*. Thereby, it is proved, by induction over the size of the topology, that any property that holds for a model of a system with a star topology will hold for a model of any system size and configuration. The proof relies on showing that any path in a model of the system of a given size is ‘equivalent’ to a path in a model of a smaller system. Thus, a relationship is established between any system model and a set of models of smaller systems. Hence, by induction the proof goes through.

We extend this approach to randomised distributed systems and generalise the proof for *any probabilistic* system that can be shown to be *degenerative.* Informally, a distributed system is degenerative if it is guaranteed that, for *some* subset of the processes, each process *degenerates.* Processes degenerate if they reach a *degenerate* state such that the execution of the process from that point onwards does not influence the global execution of the system. For example, a group of processes might terminate or may become *quiescent*, remaining active but only re-transmitting received messages. Thus the *observed* behaviour of the system will be equivalent in some sense to that of a smaller system and so we can analyse the smaller system to determine properties of the larger one. In fact, due to the scheduling of processes, we cannot be sure which processes will degenerate and therefore we have to reason about a *set* of smaller systems in order to determine properties of the larger one.

We formalise this idea, presenting a definition of a *degenerative* family of models parameterised by a *communication topology* (represented by a family of graphs) with a *base* set of models. The definition is based on establishing equivalence between any behaviour of a model of the system of a given size and behaviours in a model of a smaller system. We then show, by induction over the topology of a system, that if a QLTL property of a certain form holds for all the base models then it will hold for any model in the family. This forms the basis of an induction proof schema for reasoning about degenerative parameterised models of a system over a communication topology.

Note that, rather than introduce all the aspects of our technique simultaneously, we present a series of theorems, each extending the next, using two case studies to demonstrate some facets of the problem that we are considering. In particular, in this chapter we consider *deterministically* degenerative systems. Intuitively, a system degenerates deterministically if, under some scheduling of the processes a particular set of processes degenerate with probability 1. Thus the probabilistic element does not influence the degenerative behaviour of the system in question. We extend this definition to *probabilistically* degenerative systems in Chapter 6. Whereas for a deterministically degenerative system, under some scheduling, we know exactly which set of processes will degenerate, for a probabilistically degenerative system we only know that with probability 1 some set of processes will degenerate. Therefore, we could have different sets of processes degenerating with different probabilities but the sum of these probabilities equals 1.

For our case study of deterministically degenerative systems we consider a family of models of the IEEE 1394 Firewire tree identify protocol [33] and provide an inductive proof over the system topology (trees) that a certain class of probabilistic temporal logic properties that are satisfied by a model of a system with a star topology are satisfied by a model of a system with any acyclic topology. In Chapter 6 we consider a probabilistically degenerative system, namely the Itai Rodeh leader election protocol for rings and prove a similar result, this time over the set of rings. The examples we consider are well-understood protocols that have been analysed and verified using a variety of methods, including model-checking. We can therefore compare the results of applying our technique to these methods and demonstrate the advantages and weaknesses of our approach. In achieving the same results as these other methods, we can also have a greater degree of confidence in the correctness and applicability of our technique.

5.2 Communication Topologies and Graph Reductions

As for our approach detailed in Chapter 4, much of the work on parameterised verification of distributed systems assumes a regular topology (e.g. a star or a ring), so that the system is parameterised by the number of processes (but not always, see Balaban et al. [12]). In this chapter we do not make this presumption, but instead consider irregular topologies (although we do assume that the topology satisfies certain properties). In order to describe our systems we therefore need to formalise the notion of a topology, which we do using graphs.

For a detailed account of graph theoretic terms and notation see, for example, [15]. We give a brief outline of those necessary for our definition of a topology. A *vertex-labelled graph*, $\mathcal{G} = (E, V, I)$, is a tuple where V is a set of *vertices*, E is a set of *edges* between pairs of vertices and I is a *labelling* of vertices with each vertex $v \in V$ *uniquely* labelled by a value $I(v) \in \{0, 1, \ldots, |\mathcal{G}| - 1\}$ (where $|\mathcal{G}| = |V|$ is the *size* of the graph). A *directed* graph is a graph such that the pairs of edges are ordered i.e. we distinguish between the edge (v, w) and the edge (w, v). A *simple* directed graph does not have any multiple edges (two or more edges connecting the same two vertices) or loops (edges from a vertex to itself). A *non-trivial* graph is a graph with at least two vertices and one edge. A *connected* graph is a graph where every pair of vertices is connected by a path (a sequential set of set of edges). A graph is *finite* if the set of vertices is finite.

Definition 5.2.1. *A communication graph is a vertex-labelled, non-trivial, directed, finite, simple, connected graph.*

Since we do not consider any other type of graph, in this chapter we will often refer to a communication graph simply as a graph. Also, for a communication graph $\mathcal{G} = (E, V, I)$ we let i denote the vertex v with $I(v) = i$ and we refer to v as process i. If an edge $(v, w) \in E$, with $I(v) = i$ and $I(w) = j$ we say that process i and process j communicate. We can now define a communication topology in a straightforward manner.

Definition 5.2.2. *A communication topology (or simply a topology) is a set of communication graphs.*

In the introduction we informally described a system as degenerative if it eventually behaves as a 'smaller' system. We formalise the notion of 'smaller' in terms of the communication topology of a system. Before doing so we give some standard definitions.

Definition 5.2.3. *For a communication graph, $\mathcal{G} = (E, V, I)$ and $V' \subseteq V$, $\mathcal{G}[V'] = (E', V', I')$ is the subgraph induced by V' where $(v, w) \in E'$ iff $(v, w) \in E$ and $v, w \in V'$ and I' is the labelling I restricted to the vertices in V'.*

Definition 5.2.4. *Let $\mathcal{G} = (E, V, I)$ be a communication graph with, for all $v \in V$, $I(v)$ unique from $\{0, 1, \ldots, |\mathcal{G}| - 1\}$. Given a permutation σ on $\{0, 1, \ldots, |\mathcal{G}| - 1\}$, the permuted graph under σ is defined as $\sigma(\mathcal{G}) = (E, V, I')$ where $I'(v) = \sigma(I(v))$. Furthermore, σ is a permutation on $\mathcal{G}$.*

Based on these two definitions, we can now define a reduced graph.

Definition 5.2.5. *Let Γ be a communication topology and let $\mathcal{G} = (E, V, I) \in \Gamma$. Let σ be a permutation of the set of vertex labels of $\mathcal{G}$ and let $W \subset V$. Then $\mathcal{R} = (W, \sigma)$ is a complete reduction of $\mathcal{G}$ in Γ if and only if the graph $\mathcal{R}(\mathcal{G}) = \sigma(\mathcal{G})[W]$ belongs to Γ. We* *describe $\mathcal{R}(\mathcal{G})$ as the reduced communication graph of $\mathcal{G}$ in Γ under $\mathcal{R}$ or simply a reduced communication graph of $\mathcal{G}$.*

In the remainder of this chapter, since we only consider reductions that are complete we often refer to complete reductions simply as reductions.

Complete reductions relate communication graphs within a topology in such a way that we can consider a communication graph $\mathcal{G}$ to be larger than a reduced graph of $\mathcal{G}$. We also want to establish a set of 'least' elements of a communication topology. We can do this by showing that we can reduce a graph (under some sequence of reductions) in the topology until we have a graph that belongs to some subset of the communication topology.

Definition 5.2.6. *Let Φ and Γ be communication topologies such that $\Phi \subset \Gamma$ and let $\mathcal{Q}_\Gamma = \{Q_\mathcal{G} | \mathcal{G} \in \Gamma\}$ be a family of sets of complete reductions for communication graphs in Γ such that for all $\mathcal{G} \in \Phi$, $\mathcal{Q}_\mathcal{G} = \emptyset$. Then Γ is reducible to Φ under $\mathcal{Q}_\Gamma$ if and only if, for all $\mathcal{G} \in \Gamma \setminus \Phi$, there exists a sequence of reductions, $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n$ (for some $n \geq 1$) such that, for all $1 \leq i \leq n$,*

$$\mathcal{R}_i \in Q_{\mathcal{R}_{i-1}(\mathcal{R}_{i-2}(\ldots(\mathcal{R}_1(\mathcal{G}))))} \text{ and,}$$

$$\mathcal{R}_n(\mathcal{R}_{n-1}(\ldots(\mathcal{R}_1(\mathcal{G}))\ldots)) \in \Phi.$$

5.3 Specifying Parameterised Sets of Models

Our technique is geared towards the use of model checking tools as a means for modelling and understanding (and verifying, for small instances) the behaviour of parameterised probabilistic concurrent systems. Therefore we consider (MDP) models of distributed systems derived from specifications given in high-level modular modelling languages (e.g. Probmela [7] or PRISM [40]) whereby each process in a system is defined as a module, and is isomorphic to every other process, up to communication and process index.

We assume that the specifications are described in terms of variable sets that are partitioned according to *local*, *global* and *channel* variables, where the local variables, defined within a module, are the same (up to indexing) for each process and can only be examined/updated by commands in that module. The global variables are common to all processes and the channel variables are used to send messages between a pair of processes. Actions are local to a process, and are therefore indexed (by processes).

Since we are considering models parameterised by a communication topology, we need to consider specifications defined over communication graphs. We assume that a module is defined for every process of a communication graph and that the channel variables are defined according to the processes that can communicate. Formally,

Definition 5.3.1. *Given a communication graph, $\mathcal{G} = (E, V, I)$ ($|\mathcal{G}| = N$), $\mathcal{S}(\mathcal{G})$ is a specification over $\mathcal{G}$ if it has variable set, $X(\mathcal{S}(\mathcal{G})) = \cup_{i=0}^{N-1} X^i \cup G \cup C$ and action set $Act(\mathcal{S}(\mathcal{G})) = \cup_{i=0}^{N-1}$ such that for $0 \leq i \leq N-1$,*

- *$X^i = \{x_1^i, x_2^i, \ldots, x_m^i\}$ is a set of* local *variables associated with process i, with domains $D(x_j^i)$ ($1 \leq j \leq m$),*
- *$G = \{g_1, g_2, \ldots, g_n\}$ is a set of* global *variables with domains $D(g_j)$ ($1 \leq j \leq n$),*
- *$C = \{c_{j,k} | (j,k) \in E\}$ is a set of* channel *variables with domains $D(c_{j,k})$ where $\forall c_{j,k}, c_{j',k'} \in C$, $D(c_{j,k}) = D(c_{j',k'})$,*
- *$Act_i = \{a_1^i, a_2^i, \ldots, a_l^i\}$ is a (possibly empty) set of* local *actions associated with process i.*

Definition 5.3.2. *Given a communication topology, Γ, a parameterised specification, $\mathcal{S}(\Gamma) = \{\mathcal{S}(\mathcal{G}) | \mathcal{G} \in \Gamma\}$, over Γ is a set of specifications over the communication graphs of Γ.*

Specifications are given in a high-level modelling language. However, in order to define a degenerative system we need to reason at the level of MDPs, and so we need to consider the models derived from a specification. Note that this is dependent on the semantics of the modelling language employed: we give a generic definition below.

Definition 5.3.3. *Let $\mathcal{G} = (E, V, I)$ ($|\mathcal{G}| = N$) be a communication graph and let $\mathcal{S}(\mathcal{G})$ be a specification over $\mathcal{G}$. Let the initial value of the variables in $X(\mathcal{S}(\mathcal{G}))$ be given by $init(X(\mathcal{S}(\mathcal{G})))$ and let the model of $\mathcal{S}(\mathcal{G})$ be a MDP,*

$$\mathcal{M}(\mathcal{S}(\mathcal{G})) = (D(X(\mathcal{S}(\mathcal{G}))), init(X(\mathcal{S}(\mathcal{G}))), Steps(\mathcal{S}(\mathcal{G})), Act(\mathcal{S}(\mathcal{G})), L)$$

such that L labels states with subsets of the set of atomic propositions AP defined over $X(\mathcal{S}(\mathcal{G}))$ and $Steps(\mathcal{S}(\mathcal{G}))$ is defined over the set of updates of the specification according to the semantics of the specification language.

Note that in the models we consider the set of states as tuples over the possible values of the variables of the specification that the model is derived from. For a graph $\mathcal{G}$ and a specification $\mathcal{S}(\mathcal{G})$ over $\mathcal{G}$ let $X(\mathcal{S}(\mathcal{G})) = \cup_{i=0}^{N-1} X^i \cup G \cup C$ be the corresponding set of variables. Then if $X^i = \{x_1^i, x_2^i, \ldots, x_m^i\}$, we let $\bar{X}^i$ denote the tuple $(x_1^i, x_2^i, \ldots, x_m^i)$, and similarly for $\bar{G}$ and $\bar{C}$. Then the tuple $\bar{X}(\mathcal{S}(\mathcal{G}))$ is given by $(\bar{X}^0, \bar{X}^1, \ldots, \bar{X}^{N-1}, \bar{G}, \bar{C})$.

Definition 5.3.4. *For a communication topology,* Γ*, let* $\mathcal{S}(\Gamma)$ *be a parameterised specification over* Γ*. Then* $\mathbf{M}(\mathcal{S}(\Gamma)) = \{\mathcal{M}(\mathcal{S}(\mathcal{G})) | \mathcal{S}(\mathcal{G}) \in \mathcal{S}(\Gamma)\}$ *is the parameterised family of models over* $\mathcal{S}(\Gamma)$*.*

We will often abbreviate a model $\mathcal{M}(\mathcal{S}(\mathcal{G}))$ to $\mathcal{M}_{\mathcal{G}}$ and a parameterised family of models, $\mathbf{M}(\mathcal{S}(\Gamma))$ to $\mathbf{M}_\Gamma$.

5.3.1 Indexed Variables

When establishing properties of a set of models that are degenerative we will need to consider reductions of graphs and so we will need to define permutations of process indices. Hence, the properties that we analyse cannot refer to the indices of a process. We therefore distinguish between *indexed* and *unindexed* variables and variable domains. Note that this is an imposed distinction. In general specifications do not explicitly consider these types of variables separately.

Let $\mathcal{G}$ be a graph and $\mathcal{S}(\mathcal{G})$ a specification over $\mathcal{G} = (E, V, I)$ ($|\mathcal{G}| = N$) with variable set $X(\mathcal{S}(\mathcal{G})) = \cup_{i=0}^{N-1} X^i \cup G \cup C$. Let $I(V) = \{I(v) | v \in V\}$ denote the *index set* of $\mathcal{S}(\mathcal{G})$. For $i \in I(V)$, we say that a variable $x_j^i \in X^i$ is indexed and has index i. If there is a set of global variables $g^0, g^1, \ldots, g^{N-1}$ with identical domains we also say that g^i is indexed and has index i. Note also that all channel variables are indexed by two process indices. For example, channel $c_{i,j}$ has indices i and j.

Furthermore, for a local or global variable x of $X(\mathcal{S}(\mathcal{G}))$, we say that the domain of x is indexed if the domain is defined by $D(x) = I(V) \cup \{\perp\}$, where $\perp$ represents an unassigned value. If x has an indexed domain and $x = i$ then x is indexed and has index value i. Note that we assume that channels do not have indexed domains. A variable is unindexed if it is not indexed and does not have an indexed domain.

We can extend the definition of indexed variables to indexed propositions. For the set of atomic propositions AP over $X(\mathcal{S}(\mathcal{G}))$, we say that a proposition $x = v$ for $x \in X$, $v \in D(x)$ is indexed if x is indexed or x has an indexed domain and $v \neq \perp$, otherwise $x = v$ is an unindexed proposition. Furthermore, for a QLTL or PCTL property ϕ with atomic propositions over AP, ϕ is an unindexed property if it contains only unindexed propositions.

5.4 Mappings Induced by a Permutation of a Communication Graph

The reduction of a communication graph requires permutation of the graph as well as removal of vertices. In order to show that a set of MDPs parameterised by a topology is degenerative we establish that the model of the specification of a communication graph behaves in the same manner as the model of the specification of the permuted graph. We therefore give a series of definitions of mappings induced by a permutation that allows us to establish isomorphism between adversaries of these models (see Section 3.5.3).

Given a specification over a graph and a specification over a permutation of the graph, if the specifications are equivalent up to process indices i.e. the variables (excluding the channel variables), variable domains and action sets are the same, then we can define a mapping between the state spaces of the models of the specifications that permutes process indices according to the graph permutation.

Definition 5.4.1. *Let $\mathcal{G}$ ($|\mathcal{G}| = N$) be a communication graph and $\mathcal{S}(\mathcal{G})$ a specification over $\mathcal{G}$. Let $\sigma : \{\perp, 0, \ldots, N-1\} \to \{\perp, 0, \ldots, N-1\}$ be a permutation of $\mathcal{G}$ (such that $\sigma(\perp) = \perp$) and let $\mathcal{S}(\sigma(\mathcal{G}))$ be a specification over $\sigma(\mathcal{G})$ such that,*

$$\begin{aligned} X(\mathcal{S}(\mathcal{G})) \setminus C_{\mathcal{G}} &= X(\mathcal{S}(\sigma(\mathcal{G}))) \setminus C_{\sigma(\mathcal{G})}, \\ D(X(\mathcal{S}(\mathcal{G}))) &= D(X(\mathcal{S}(\sigma(\mathcal{G})))), \\ Act(\mathcal{S}(\mathcal{G})) &= Act(\mathcal{S}(\sigma(\mathcal{G}))). \end{aligned}$$

Let $\mathcal{M}_{\mathcal{G}} = (S, s_0, Steps, Act, L)$ and $\mathcal{M}_{\sigma(\mathcal{G})} = (S', s'_0, Steps', Act', L')$ be the MDPs over $\mathcal{S}(\mathcal{G})$ and $\mathcal{S}(\sigma(\mathcal{G}))$, respectively.

The index map on S induced by σ, $\varsigma : S \to S'$, is defined as follows. For any state $s = (\bar{X}_0, \bar{X}_1, \ldots, \bar{X}_{N-1}, \bar{G}, \bar{C}) \in S$, define $\varsigma(s) = (\varsigma(\bar{X}_0), \varsigma(\bar{X}_1), \ldots, \varsigma(\bar{X}_{N-1}), \varsigma(\bar{G}), \varsigma(\bar{C}))$

such that,

$\forall 0 \leq i \leq N-1, \varsigma(\bar{X}_i) = (\varsigma(x_i^1), \varsigma(x_i^2), \ldots, \varsigma(x_i^m))$, *where for* $x_i^j \in X_i$,

$$\varsigma(x_i^j) = \begin{cases} \sigma(x^j_{\sigma^{-1}(i)}) & \textit{if } x_i^j \textit{ has an indexed domain} \\ x^j_{\sigma^{-1}(i)} & \textit{otherwise} \end{cases}$$

and $\varsigma(\bar{G}) = (\varsigma(g^1), \varsigma(g^2), \ldots, \varsigma(g^m))$, *where for* $g^j \in G$,

$$\varsigma(g^j) = \begin{cases} \sigma(g^j_{\sigma^{-1}(i)}) & \textit{if } g^j \textit{ has an indexed domain and is indexed with } g^j = g_i^j \\ g^j_{\sigma^{-1}(i)} & \textit{if } g^j \textit{ is indexed with } g^j = g_i^j \\ \sigma(g^j) & \textit{if } g^j \textit{ has an indexed domain} \\ g^j & \textit{otherwise} \end{cases}$$

and $\varsigma(\bar{C}) = \sigma^{-1}(\bar{C})$ *where we let* $\sigma^{-1}(\bar{C})$ *denote the set of channel values obtained such that the value of* $c_{i,j}$ *in* $\sigma^{-1}(\bar{C})$ *is the value of channel* $c_{\sigma^{-1}(i),\sigma^{-1}(j)}$ *in* C.

Similarly, we can define a mapping over the atomic propositions over the variable set of a specification.

Definition 5.4.2. *Let* $\mathcal{G}$ *(*$|\mathcal{G}| = N$*) be a communication graph and let* $\mathcal{S}(\mathcal{G})$ *be a specification over* $\mathcal{G}$*. Let* $\sigma : \{\perp, 0, \ldots, N-1\} \rightarrow \{\perp, 0, \ldots, N-1\}$ *be a permutation of* $\mathcal{G}$ *(such that* $\sigma(\perp) = \perp$*) and let* $\mathcal{S}(\sigma(\mathcal{G}))$ *be a specification over* $\sigma(\mathcal{G})$ *such that*

$$X(\mathcal{S}(\mathcal{G})) = X(\mathcal{S}(\sigma(\mathcal{G}))), D(X(\mathcal{S}(\mathcal{G}))) = D(X(\mathcal{S}(\sigma(\mathcal{G})))), Act(\mathcal{S}(\mathcal{G})) = Act(\mathcal{S}(\sigma(\mathcal{G}))).$$

Let AP *be the set of propositions over* $X(\mathcal{S}(\mathcal{G}))$ *and let* AP' *be the set of propositions over* $X(\mathcal{S}(\sigma(\mathcal{G})))$*. The index proposition map induced by* σ*,* $\Sigma : AP \rightarrow AP'$*, is defined as follows.*

For an unindexed proposition $x = v$*, where* $x \in X(\mathcal{S}(\mathcal{G}))$*,* $v \in D(x)$ *and* x *is unindexed,*

$$\Sigma(x = v) = x = v.$$

For an indexed proposition $x_i = v$*, where* $x \in X(\mathcal{S}(\mathcal{G}))$ *is indexed by* i*, but* $D(x)$ *is unindexed,*

$$\Sigma(x_i = v) = x_{\sigma(i)} = v.$$

For an indexed proposition $x_i = v$*, where* $x \in X(\mathcal{S}(\mathcal{G}))$ *is indexed by* i*, and* $D(x)$ *is an indexed domain,*

$$\Sigma(x_i = v) = x_{\sigma(i)} = \sigma(v).$$

For an indexed proposition $x = v$, where $x \in X(\mathcal{S}(\mathcal{G}))$ is unindexed, and $D(x)$ is an indexed domain,

$$\Sigma(x = v) = x = \sigma(v).$$

We define degenerative behaviour in terms of the adversaries of a model. Therefore, we extend the index map induced by a permutation, defined over states, to the path index map induced by a permutation, which we define over finite paths of a model.

Definition 5.4.3. *Let $\mathcal{G}$ ($|\mathcal{G}| = N$) be a communication graph and let $\mathcal{S}(\mathcal{G})$ be a specification over $\mathcal{G}$. Let $\sigma : \{\perp, 0, \ldots, N-1\} \to \{\perp, 0, \ldots, N-1\}$ be a permutation of $\mathcal{G}$ (such that $\sigma(\perp) = \perp$) and let $\mathcal{S}(\sigma(\mathcal{G}))$ be a specification over $\sigma(\mathcal{G})$ such that*

$$X(\mathcal{S}(\mathcal{G})) = X(\mathcal{S}(\sigma(\mathcal{G}))), D(X(\mathcal{S}(\mathcal{G}))) = D(X(\mathcal{S}(\sigma(\mathcal{G})))), Act(\mathcal{S}(\mathcal{G})) = Act(\mathcal{S}(\sigma(\mathcal{G}))).$$

Let $\mathcal{M}_{\mathcal{G}} = (S, s_0, Steps, Act, L)$ and $\mathcal{M}_{\sigma(\mathcal{G})} = (S', s'_0, Steps', Act', L')$ be the MDPs over $\mathcal{S}(\Gamma)$ and $\mathcal{S}(\sigma(\Gamma))$ respectively and let AP, AP' be the atomic propositions over $X(\mathcal{S}(\mathcal{G}))$, $X(\mathcal{S}(\sigma(\mathcal{G})))$ respectively.

Let ς be the index map on S induced by σ and A and A' adversaries of $\mathcal{M}$ and $\mathcal{M}'$ respectively. The path index map induced by σ is defined by $\rho : Path^{A}_{fin}(s_0) \to Path^{A'}_{fin}(s'_0)$ such that, for $n \geq 0$,

$$\rho(\omega) = \varsigma(s_0) \xrightarrow{(a_0^{\sigma(i_0)}, \mu'_0)} \varsigma(s_1) \xrightarrow{(a_1^{\sigma(i_1)}, \mu'_1)} \ldots \xrightarrow{(a_{n-1}^{\sigma(i_{n-1})}, \mu'_{n-1})} \varsigma(s_n),$$

where,

$$\omega = s_0 \xrightarrow{(a_0^{i_0}, \mu_0)} s_1 \xrightarrow{(a_1^{i_1}, \mu_1)} \ldots \xrightarrow{(a_{n-1}^{i_{n-1}}, \mu_{n-1})} s_n \in Path^{A}_{fin}(s_0).$$

If this mapping induces an isomorphism between models then the probabilities over LTL path formula are preserved.

Lemma 5.4.4. *Let $\mathcal{G}$ ($|\mathcal{G}| = N$) be a communication graph and let $\mathcal{S}(\mathcal{G})$ be a specification over $\mathcal{G}$. Let $\sigma : \{\perp, 0, \ldots, N-1\} \to \{\perp, 0, \ldots, N-1\}$ be a permutation of $\mathcal{G}$ (such that $\sigma(\perp) = \perp$) and let $\mathcal{S}(\sigma(\mathcal{G}))$ be a specification over $\sigma(\mathcal{G})$ such that*

$$X(\mathcal{S}(\mathcal{G})) = X(\mathcal{S}(\sigma(\mathcal{G}))), D(X(\mathcal{S}(\mathcal{G}))) = D(X(\mathcal{S}(\sigma(\mathcal{G})))), Act(\mathcal{S}(\mathcal{G})) = Act(\mathcal{S}(\sigma(\mathcal{G}))).$$

Let $\mathcal{M}_{\mathcal{G}} = (S, s_0, Steps, Act, L)$ and $\mathcal{M}_{\sigma(\mathcal{G})} = (S', s'_0, Steps', Act', L')$ be the MDPs associated with $\mathcal{S}(\Gamma)$ and $\mathcal{S}(\sigma(\Gamma))$ respectively and let AP and AP' be the atomic propositions over $X(\mathcal{S}(\mathcal{G}))$, $X(\mathcal{S}(\sigma(\mathcal{G})))$ respectively.

Let ς be the index map on S induced by σ, Σ the proposition index map induced by σ and A and A' adversaries of $\mathcal{M}$ and $\mathcal{M}'$ respectively and let ρ be the path index map on A induced by σ. Suppose that ρ is an isomorphism between D^A and $D^{A'}$, the DTMCs induced by A and A' respectively such that $\rho(s_0) = s'_0$. Then, for $s \in S$, and LTL path formula, ψ,

$$Prob^A_s(\{\omega \in Path^A(s) | \omega \models \psi\}) = Prob^{A'}_{\varsigma(s)}(\{\omega' \in Path^{A'}(\varsigma(s)) | \omega' \models \Sigma(\psi)\})$$

Proof. Let $s \in S$. For $\omega \in Path^A(s)$ and atomic proposition $a \in AP$, $a \in L(last(\omega))$ if and only if $\Sigma(a) \in L'(last(\rho(\omega)))$. Hence, since ρ is an isomorphism it follows from Lemma 3.5.17 that,

$$Prob^A_s(\{\omega \in Path^A(s) | \omega \models \psi\}) = Prob^{A'}_{\varsigma(s)}(\{\omega' \in Path^{A'}(\varsigma(s)) | \omega' \models \Sigma(\psi)\}).$$

□

5.5 Deterministically Degenerative Parameterised Sets of Models

We now turn to our main definition that gives conditions for a parameterised family of models (over a parameterised specification for some topology) to be *degenerative*. The key condition is that the communication graphs of the topology are reduced such that every adversary of a model of a specification of some graph is stuttering equivalent to an adversary of a model of a specification over a reduced graph.

Definition 5.5.1. *Let Γ be a communication topology that is reducible under a family of reductions $\mathcal{Q}_\Gamma = \{Q_\mathcal{G} | \mathcal{G} \in \Gamma\}$. Furthermore, suppose $\mathcal{S}(\Gamma)$ is a parameterised specification over Γ, and let*

$$\mathbf{M}_\Gamma = \{\mathcal{M}_\mathcal{G} = (S_\mathcal{G}, s^\mathcal{G}_0, Steps_\mathcal{G}, Act_\mathcal{G}, L_\mathcal{G}) | \mathcal{G} \in \Gamma\},$$

be the parameterised family of models over $\mathcal{S}(\Gamma)$.

For each $\mathcal{G} \in \Gamma$ let $X_\mathcal{G}$ be the set of variables of $\mathcal{S}(\mathcal{G})$ (with $C_\mathcal{G} \subseteq X_\mathcal{G}$ the set of channel variables) and let $AP_\mathcal{G}$ be the atomic propositions over $X_\mathcal{G}$. For each $\mathcal{G} \in \Gamma$ and each $\mathcal{R} \in Q_\mathcal{G}$, define a set of variables $X'_{\mathcal{R}(\mathcal{G})} \subseteq X_{\mathcal{R}(\mathcal{G})}$ (with $AP'_{\mathcal{R}(\mathcal{G})} \subseteq AP_{\mathcal{R}(\mathcal{G})}$, the set of atomic propositions over $X'_{\mathcal{R}(\mathcal{G})}$). Then $\mathbf{M}_\Gamma$ is deterministically degenerative *with* base Φ *under $\mathcal{Q}_\Gamma$ if and only if,*

1. ***(Reduced Variables and Actions:)*** *For $\mathcal{G} \in \Gamma$ and a reduction $\mathcal{R} = (W, \sigma) \in Q_{\mathcal{G}}$,*

$$X_{\sigma(\mathcal{G})} \setminus C_{\mathcal{G}} = X_{\mathcal{G}} \setminus C_{\mathcal{G}}, D(X_{\sigma(\mathcal{G})}) = D(X_{\mathcal{G}}), Act_{\sigma(\mathcal{G})} = Act_{\mathcal{G}},$$

$$X_{\mathcal{R}(\mathcal{G})} \subseteq X_{\sigma(\mathcal{G})}, D(X_{\mathcal{R}(\mathcal{G})}) \subseteq D(X_{\sigma(\mathcal{G})}), Act_{\mathcal{R}(\mathcal{G})} \subseteq Act_{\sigma(\mathcal{G})},$$

2. ***(Matching Adversaries:)*** *For $\mathcal{G} \in \Gamma \setminus \Phi$, there exists $\mathcal{R} = (W, \sigma) \in Q_{\mathcal{G}}$ such that, for every adversary A of $\mathcal{M}_{\mathcal{G}}$, there exists an adversary A' of $\mathcal{M}_{\sigma(\mathcal{G})}$ that is isomorphic to A under the path index map induced by σ, with A' stuttering equivalent to some adversary A'' of $\mathcal{M}_{\mathcal{R}(\mathcal{G})}$ with respect to $AP'_{\mathcal{R}(\mathcal{G})}$.*

The establishment of a set of models, parameterised by a set of topologies, that is degenerative provides an inductive basis (over the set of topologies) with which to establish properties of the models. The proof of Theorem 5.5.2 is a generalisation of a proof for a specific instance of a degenerative non-probabilistic system [49].

Theorem 5.5.2. *Let Γ be a set of communication topologies that is reducible to Φ under the family of sets of reductions, $\mathcal{Q}_{\Gamma}$, and let $\mathcal{S}(\Gamma)$ be a set of specifications over Γ. Suppose that for each $\mathcal{G} \in \Gamma$, $\mathcal{R} \in Q_{\mathcal{G}}$, there is a set of variables $X'_{\mathcal{R}(\mathcal{G})} \subseteq X_{\mathcal{R}(\mathcal{G})}$ (with $AP'_{\mathcal{R}(\mathcal{G})} \subseteq AP_{\mathcal{R}(\mathcal{G})}$, the set of atomic propositions over $X'_{\mathcal{R}(\mathcal{G})}$) such that the family of models over $\mathcal{S}(\Gamma)$, $\mathbf{M}_{\Gamma}$, is deterministically degenerative with base Φ under $\mathcal{Q}_{\Gamma}$. Then for any unindexed $QLTL_{\setminus\mathcal{X}}$ property ϕ with atomic propositions in $\bigcap_{\mathcal{G}\in\Gamma} \bigcap_{(W,\sigma)\in Q_{\mathcal{G}}} AP'_{\mathcal{R}(\mathcal{G})}$, if $\mathcal{M}_{\mathcal{F}} \models \phi$ for all $\mathcal{F} \in \Phi$, $\mathcal{M}_{\mathcal{G}} \models \phi$ for all $\mathcal{G} \in \Gamma$.*

Proof. Let $\mathcal{S}(\Gamma)$ be a parameterised specification over Γ, a communication topology. Suppose that there exists a parameterised set of models over $\mathcal{S}(\Gamma)$,

$$\mathbf{M}_{\Gamma} = \{\mathcal{M}_{\mathcal{G}} | \mathcal{G} \in \Gamma\},$$

that is deterministically degenerative with base Φ, under $\mathcal{Q}_{\Gamma} = \{Q_{\mathcal{G}}\}$, a family of sets of reductions such that Γ is reducible to Φ under $\mathcal{Q}_{\Gamma}$. For each $\mathcal{G} \in \Gamma$, let the atomic propositions $AP_{\mathcal{G}}$ be defined over $X_{\mathcal{G}} = X(\mathcal{S}(\mathcal{G}))$.

For each $\mathcal{G} \in \Gamma \setminus \Phi$ and $\mathcal{R} = (W, \sigma) \in Q_{\mathcal{G}}$, let $X'_{\mathcal{R}(\mathcal{G})} \subseteq X_{\mathcal{R}(\mathcal{G})}$ be a set of variables such that Condition 2 of Definition 5.5.1 holds. Let $AP'_{\mathcal{R}(\mathcal{G})} \subseteq AP_{\mathcal{R}(\mathcal{G})}$, be the set of atomic propositions over $\mathbf{X}'_{\mathcal{R}(\mathcal{G})}$.

Let $\mathcal{G} \in \Gamma$ and suppose that ϕ is an unindexed QLTL$_{\backslash\mathcal{X}}$ property with atomic propositions in $\bigcap_{\mathcal{R}\in Q_{\mathcal{G}}} AP'_{\mathcal{R}(\mathcal{G})}$. Assume that $\mathcal{M}_{\mathcal{R}(\mathcal{G})} \models \phi$, for every $\mathcal{R} \in Q^{\mathcal{G}}$. Then we can show that $\mathcal{M}_{\mathcal{G}} \models \phi$ as follows.

Let $A \in Adv_{\mathcal{M}_{\mathcal{G}}}$. Choose $\mathcal{R} = (W, \sigma) \in Q_{\mathcal{G}}$ such that A is isomorphic to some adversary $A' \in Adv_{\mathcal{M}_{\sigma(\mathcal{G})}}$ under the path index map induced by σ, ρ, with A' stuttering equivalent to some adversary $A'' \in Adv_{\mathcal{M}_{\mathcal{R}(\mathcal{G})}}$ w.r.t. $AP'_{\mathcal{R}(\mathcal{G})}$.

The property ϕ has the form $\mathcal{P}_{\bowtie p}[\psi]$. If Σ is the proposition index map induced by σ then $\Sigma(\psi) = \psi$ since ϕ is unindexed. For every adversary B of $\mathcal{M}_{\mathcal{R}(\mathcal{G})}$,

$$Prob^{B}_{s''_0}(\{\omega'' \in Path^{B}_{s''_0} | \omega'' \models \psi\}) \bowtie p. \tag{5.1}$$

If $\mathcal{M}_{\mathcal{G}}$, $\mathcal{M}_{\sigma(\mathcal{G})}$ and $\mathcal{M}_{\mathcal{R}(\mathcal{G})}$ have initial states s_0, s'_0 and s''_0 respectively then,

$$\begin{aligned}
&Prob^{A}_{s_0}(\{\omega \in Path^{A}_{s_0} | \omega \models \psi\}) \\
&= Prob^{A'}_{s'_0}(\{\omega' \in Path^{A'}_{s'_0} | \omega' \models \psi\})\text{by Lemma 5.4.4 since } A = A' \text{ under } \rho \\
&= Prob^{A''}_{s''_0}(\{\omega'' \in Path^{A''}_{s''_0} | \omega'' \models \psi\}) \text{ since } A' \simeq A'' \text{ w.r.t. } AP'_{\mathcal{R}(\mathcal{G})} \\
&\bowtie p \text{ by 5.1.}
\end{aligned}$$

Since the above is true for every adversary of $\mathcal{M}_{\mathcal{G}}$, $\mathcal{M}_{\mathcal{G}} \models \phi$.

Let ϕ be an unindexed QLTL$_{\backslash\mathcal{X}}$ formula with atomic propositions in $\bigcap_{\mathcal{G}\in\Gamma} \bigcap_{\mathcal{R}\in Q_{\mathcal{G}}} AP'_{\mathcal{R}(\mathcal{G})}$ and let $\mathcal{G} \in \Gamma$. If $\mathcal{G} \in \Phi$ then, by the statement of the theorem, $\mathcal{M}_{\mathcal{G}} \models \phi$.

Assume that $\mathcal{G} \in \Gamma \setminus \Phi$. Since ϕ has atomic propositions in $\bigcap_{\mathcal{G}\in\Gamma} \bigcap_{(W,\sigma)\in Q_{\mathcal{G}}} AP'_{\mathcal{R}(\mathcal{G})}$, ϕ is defined over $\cap_{\mathcal{R}\in Q_{\mathcal{G}}} AP'_{\mathcal{R}(\mathcal{G})}$. Since ϕ is also unindexed, by the above, $\mathcal{M}_{\mathcal{G}} \models \phi$ if $\mathcal{M}_{\mathcal{R}(\mathcal{G})} \models \phi$ for all $\mathcal{R} \in Q_{\mathcal{G}}$. For each $\mathcal{R} \in Q_{\mathcal{G}}$, either $\mathcal{R}(\mathcal{G})$ is in Φ or it can be reduced further. Since Γ is reducible to Φ under $\mathcal{Q}_{\Gamma}$, continuing in this way, we can construct a tree of graphs in which every terminal node is a graph in Φ. Finally, by the statement of the theorem, each of the models associated with the graphs at these terminal nodes satisfy ϕ and, by propagation up the tree of graphs, it follows that $\mathcal{M}_{\mathcal{G}} \models \phi$. $\square$

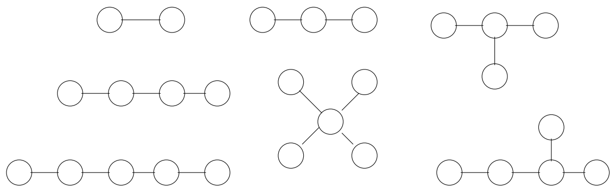


Figure 5.1: Acyclic topologies for 2, 3, 4 and 5 processes.

5.6 Case Study: The IEEE 1394 (Firewire) Tree Identify Protocol

5.6.1 Introduction

We illustrate our technique with a case study. The IEEE 1394 (Firewire) Tree Identify Protocol (TIP), as described in the IEEE standard [33], is designed to elect a leader from a set of processes arranged in an acyclic topology (Figure 5.1 shows all acyclic communication graphs for systems of N processes for $2 \leq N \leq 5$, up to the labelling of the vertices). A process may send one of three messages to a neighbouring process: *be_my_parent* (*bmp*), *be_my_child* (*bmc*) or *acknowledge* (*ack*). Any process that has received *bmp* messages from at least all but one of its neighbours responds with *bmc* messages and, if necessary, sends a *bmp* to the remaining neighbour. The neighbouring processes will send an *ack* upon receiving a *bmc*, from which point they play no further part in the protocol (and hence the protocol is degenerative). In this manner the protocol builds a spanning tree with the root process elected as leader.

Note that it is possible for two neighbouring processes to attempt to become leader by sending *bmp* requests to each other simultaneously. In order to resolve this contention, each process probabilistically chooses to wait for a long or short amount of time, before attempting to send a request again. If a process then receives a request before it has sent one, it will be elected leader. Otherwise, another contention situation ensues and the "back-off" procedure must be repeated.

Much work has been done on proving correctness of root contention in the TIP by the modelling of two contending processes [60]. Appealing to these results, in [16] the resolution of contention is modelled by a non-deterministic choice, allowing the full protocol to be modelled non-probabilistically. Non-probabilistic properties of these models for fixed

Table 5.1: Transitions in $\mathcal{M}_{\mathcal{G}}$ made by process j when it receives requests from all of its neighbours

1. Process j receives *bmp* from all its neighbours.	$(start, [0, \dots, 0], k, 0, 0)$ $[bmp]_{i_1,j}, \dots, [bmp]_{i_k,j}$ $1 \downarrow a^j$ $(child, [i_1, i_2, \dots, i_k, 0, \dots, 0], k, 0, k)$ $[]_{i_1,j}, \dots, []_{i_k,j}$
2. Process j responds to its neighbours with *bmc* requests.	$(child, [i_1, i_2, \dots, i_k, 0, \dots, 0], k, 0, k)$ $[]_{i_1,j}, \dots, []_{i_k,j}$ $1 \downarrow b^j$ $(parent, [i_1, i_2, \dots, i_k, 0, \dots, 0], k, 0, k)$ $[bmc]_{i_1,j}, \dots, [bmc]_{i_k,j}$
3. Process j receives *ack* from all its neighbours and becomes leader.	$(parent, [i_1, i_2, \dots, i_k, 0, \dots, 0], k, 0, k)$ $[ack]_{i_1,j}, \dots, [ack]_{i_k,j}$ $1 \downarrow c^j$ $(finish, [0, \dots, 0], k, 0, 0)$ $[]_{1,0}, \dots, []_{N-1,0}, elected = j$

configurations are verified using the SPIN model checker.

We consider models for the TIP in which contention is resolved probabilistically. We do not consider the real-time aspects but instead model contention with a contending process (the one with the smallest index) making a simple probabilistic choice as follows. With probability $\frac{1}{4}$, the process loses allowing the other process to send its *bmp*. With probability $\frac{1}{4}$, the process wins and transmits its request to the other contending process. With probability $\frac{1}{2}$ contention is not resolved and the process must make the probabilistic choice again. This is an abstract representation of contention with two contending processes flipping a fair coin and choosing again if they flip the same values, or deciding on a leader if they flip different values.

5.6.2 Parameterised Specifications of the TIP

We have defined a script for automatically generating PRISM specifications of the TIP for any communication graph, based on the Promela specifications of [49]. We can view this script as generating a parameterised set of specifications for the TIP system, $\mathcal{S}(\Gamma)$, over the communication topology Γ, defined as follows.

Definition 5.6.1. *Let Γ be the communication topology such that for $\mathcal{G} = (E, V, I) \in \Gamma$,*

$(v,w) \in E$ if and only if $(w,v) \in E$ and $\mathcal{G}$ does not contain any cycles.

In other words, Γ is the set of acyclic communication graphs such that communication between processes is bi-directional. In the sequel, we let $\mathbf{M}_\Gamma = \{\mathcal{M}_\mathcal{G} | \mathcal{G} \in \Gamma\}$ be the parameterised family of models over $\mathcal{S}(\Gamma)$. We give a PRISM specification of the TIP over a star topology with three processes in Appendix E and we give a description of the specifications and the models below.

Given $\mathcal{G} \in \Gamma$, with $|\mathcal{G}| = N$, for the specification $\mathcal{S}(\mathcal{G}) \in \mathcal{S}(\Gamma)$ then $\mathbf{X}_\mathcal{G} = \cup_{i=0}^{N-1} X_\mathcal{G}^i \cup G_\mathcal{G} \cup C_\mathcal{G}$ is the set of variables for $\mathcal{S}(\mathcal{G})$. For a variable *vari* of the PRISM specification for convenience we use the notation var_i. So, for $i \in \{0, \dots, N-1\}$,

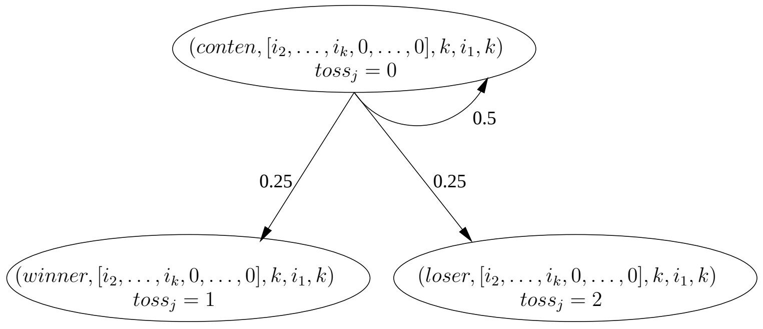
$$G_\mathcal{G} = \{elected, toss_0, toss_1, \dots, toss_{N-1}\},$$
$$C_\mathcal{G} = \{c_{g,h} | (g,h) \in E\},$$
$$X_\mathcal{G}^i = \{state_i, child_{i,0}, \dots, child_{i,N-1}, adj_i, remaining\_partner_i, no\_of\_requests_i\},$$

For $i, j \in \{0, 1, \dots, N-1\}, c_{g,h} \in C_\mathcal{G}$ the variable domains are,

$$D(state_i) = \{start, child, parent, conten, response,$$
$$complete, winner, loser, b\_child, finish\}$$
$$D(c_{g,h}) = \{empty, bmp, bmc, ack\},$$
$$D(no\_of\_requests_i) = D(adj_i) = \{0, 1, \dots, N-1\},$$
$$D(remaining\_partner_i) = D(child_{i,j}) = D(elected) = \{0, 1, \dots, N\},$$
$$D(toss_i) = \{0, 1, 2\}.$$

Note that since the channels have length one, they can be represented in PRISM by global variables. The set of actions for $\mathcal{S}(\mathcal{G})$ is given by $Act_\mathcal{G} = \cup_{i=0}^{N-1} Act_\mathcal{G}^j$ where, for $j = 0, 1, \dots, N-1$, $Act_\mathcal{G}^j$ is the set of actions for a given process, j.

We let $\mathbf{M}_\Gamma$ denote the parameterised set of models over $\mathcal{S}(\Gamma)$. From $\mathcal{S}(\Gamma)$ we can derive the set of possible transitions in a model $\mathcal{M}_\mathcal{G} \in \mathbf{M}_\Gamma$. We give a representation of these transitions (for a process, j) in Tables 5.1, 5.2 and 5.3. The sole probabilistic transition (that of resolving contention) is represented in Figure 5.2. Note that the entries in the tables and the figure represent a *set* of transitions since we only give the values of some of the global variables, the local variables of process j and the channel variables of process j. The remaining variables take a variety of values and thus the transition could take place



$(conten, [i_2, \ldots, i_k, 0, \ldots, 0], k, i_1, k)$
$toss_j = 0$
0.5
0.25
0.25
$(winner, [i_2, \ldots, i_k, 0, \ldots, 0], k, i_1, k)$
$toss_j = 1$
$(loser, [i_2, \ldots, i_k, 0, \ldots, 0], k, i_1, k)$
$toss_j = 2$

Figure 5.2: The transition in $\mathcal{M}_\mathcal{G}$ corresponding to root contention resolution between process j and process i_m ($j < i_m$).

between a number of different states. Each entry in the table corresponds to an update in the specification and is labelled by a number. We have labelled the specification in Appendix E with corresponding numbers.

The first column in the tables give an informal description of the specification update. The second column shows values of the local variables of process j along with some of the channel and global variables' values. These represent the necessary conditions for an action to occur and the result of that action. The current values of the local variables are presented as the tuple, $\bar{X}^i_\mathcal{G} = (s, [ch_0, \ldots, ch_{N-1}], a, r, n)$, representing the values of $state_j$, $child_{j,0}, \ldots, child_{j,N-1}$, adj_j, $remaining\_partner_j$, $no\_of\_requests_j$ respectively. The value of the channel variable $c_{h,i}$ is represented by $[msg]_{h,i}$ (where msg is bmp, bmc or ack) or $[]_{h,i}$ if $c_{h,i} = empty$. If a variable is not presented then its value is of no importance for that action. We assume for each table that process j has k neighbours, with indices, $i_1, i_2, \ldots, i_k$, and, for Tables 5.2 and 5.3 and Figure 5.2, that i_1 is the neighbour that does not initially send a bmp to j.

5.6.3 Model Checking the TIP

We have verified the following suite of properties for models of small, fixed configurations of the TIP system using PRISM. Note that these properties are expressed as QLTL properties with the exception of *Property 4*, which is negated outside of the $\mathcal{P}$ operator. We can still verify this by establishing that the negation of this property does *not* hold. Note also that *Property 1* is an unindexed QLTL$_{\backslash\mathcal{X}}$ property. In order to verify our properties using PRISM it was necessary to translate them into equivalent PCTL properties. Results for

Table 5.2: Transitions in $\mathcal{M}_{\mathcal{G}}$ made by process j when it receives requests from all but one of its neighbours (process i_m) – before entering contention.

4. Process j receives *bmp* from all its neighbours except i_m.	$(start, [0, \ldots, 0], k, 0, 0)$ $[bmp]_{i_1,j}, \ldots, [bmp]_{i_{m-1},j}, []_{i_m,j}, [bmp]_{i_{m+1},j}, \ldots, [bmp]_{i_k,j}$ $1 \downarrow a^j_m$ $(child, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, i_m, k)$ $[]_{i_1,j}, []_{i_2,j}, \ldots, []_{i_k,j}$
5. Process j responds with *bmc* and sends *bmp* to i_m.	$(child, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, i_m, k)$ $[]_{j,i_1}, []_{j,i_2}, \ldots, []_{j,i_k}$ $1 \downarrow b^j_m$ $(parent, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, i_m, k)$ $[bmc]_{j,i_1}, \ldots, [bmc]_{i_{m-1},j}, [bmp]_{i_m,j}, [bmc]_{i_{m+1},j}, \ldots, [bmc]_{j,i_k}$
6. Process j receives *ack* from all its child neighbours (not i_m).	$(parent, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, i_m, k)$ $[ack]_{i_1,j}, \ldots, [ack]_{i_{m-1},j}, [ack]_{i_{m+1},j}, \ldots, [ack]_{i_k,j}$ $1 \downarrow c^j_m$ $(complete, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, i_m, k)$ $[]_{i_1,j}, \ldots, []_{i_{m-1},j}, []_{i_{m+1},j}, \ldots, []_{i_k,j}$
7. Process j moves into the *response* state.	$(complete, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, i_m, k)$ $1 \downarrow d^j$ $(response, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, i_m, k)$
8. Process j receives *bmc* from its non-child neighbour (i_m).	$(response, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, i_m, k)$ $[bmc]_{i_m,j}$ $1 \downarrow e^j_m$ $(b\_child, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, 0, k)$ $[]_{i_m,j}$
9. Process j sends *ack* to its non-child neighbour (i_m) and terminates without becoming leader.	$(b\_child, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, 0, k)$ $[]_{j,i_m}$ $1 \downarrow f^j_m$ $(finish, [0, 0, \ldots, 0], k, 0, 0)$ $[ack]_{j,i_m}$
10. Process j receives *bmp* from i_m, its non-child neighbour and enters contention.	$(response, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, i_m, k)$ $[bmp]_{i_m,j}$ $1 \downarrow g^j_m$ $(conten, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, i_m, k)$ $[]_{i_m,j}$

Table 5.3: Transitions in $\mathcal{M}_{\mathcal{G}}$ made by process j when it receives requests from all but one of its neighbours (process i_m) – during and after contention.

11. Root contention ($i_m > j$)	See Figure 5.2.
12. $i_m < j$ and process j has won contention.	$(conten, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, i_m, k)$ $toss_j = 1$ $1 \downarrow u^j_m$ $(winner, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, i_m, k)$ $toss_j = 0$
13. $i_m < j$ and process j has lost contention.	$(conten, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, i_m, k)$ $toss_j = 2$ $1 \downarrow v^j_m$ $(loser, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, i_m, k)$ $toss_j = 0$
14. Process j has won contention and so sends *bmp* to i_m and returns to response state.	$(winner, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, i_m, k)$ $[]_{j,i_m}$ $1 \downarrow w^j_m$ $(response, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, i_m, k)$ $[bmp]_{j,i_m}$
15. Process j has lost contention and receives *bmp* from winning process, i_m and returns to *child* state.	$(loser, [i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_k, 0, \ldots, 0], k, i_m, k)$ $[bmp]_{i_m,j}$ $1 \downarrow x^j_m$ $(child, [i_m, 0, \ldots, 0], 1, 0, 1)$ $[]_{i_m,j}$
16. Process j has lost contention and sends *bmc* to winning process, i_m.	$(child, [i_m, 0, \ldots, 0], 1, 0, 1)$ $[]_{j,i_m}$ $1 \downarrow y^j$ $(parent, [i_m, 0, \ldots, 0], 1, 0, 1)$ $[bmc]_{j,i_m}$
17. Process j has lost contention and receives *ack* from i_m and becomes leader.	$(parent, [i_m, 0, \ldots, 0], 1, 0, 1)$ $[ack]_{i_m,j}$ $1 \downarrow z^j_m$ $(finish, [0, \ldots, 0], 1, 0, 0)$ $[]_{i_m,j}, elected = j$

Table 5.4: Statistics for PRISM models of the TIP with N processes, arranged in a star topology

N	States	Transitions	Nodes	Build time (s)	Property 1 (s)
2	63	108	1639	0.202	0.122
3	258	496	6997	2.118	0.156
4	1181	3040	31242	3.260	0.981
5	6032	20036	204825	14.293	6.600

verification of *Property 1* against models of the TIP with a star topology (with 2, 3, 4 and 5 processes) are given in Table 5.4 together with some statistics for these models.

Property 1: A leader will be elected almost surely.

$$\mathcal{P}_{\geq 1}[\mathit{true}\,\mathcal{U}\,\neg(\mathit{elected}=0)]$$

Property 2: Only one process will be elected leader almost surely.

$$\forall 0 \leq i \leq N-1 . \mathcal{P}_{\leq 0}[\Box(\mathit{elected}=i \Rightarrow (\mathit{true}\,\mathcal{U}\,(\mathit{elected}!=i)))]$$

Property 3: For all $0 \leq i \leq N-1$, process i will not remain in the start state indefinitely.

$$\mathcal{P}_{\geq 1}[\Box(\mathit{state}_i=\mathit{start} \Rightarrow (\mathit{true}\,\mathcal{U}\,(\mathit{state}_i!=\mathit{start})))]$$

Property 4: For $0 \leq i \leq N-1$, it is possible for process i to be elected leader.

$$\neg\mathcal{P}_{\leq 0}[\mathit{true}\,\mathcal{U}\,(\mathit{elected}=i)]$$

Property 5: If process i enters contention then it will be elected leader with probability at least a half $(0 \leq i \leq N-1)$.

$$\mathcal{P}_{\geq 0.5}[\Box(\mathit{state}_i=\mathit{conten} \Rightarrow (\mathit{true}\,\mathcal{U}\,(\mathit{elected}=i)))]$$

5.6.4 Parameterised Verification of the TIP

In [49] several properties of the IEEE 1394 Firewire leader election protocol are proved for the non-probabilistic case. The proof involves identifying processes of a system as *level-1* processes (processes with only one neighbouring node that is not a leaf) and using a reduction approach to compare properties of the associated model with one for a smaller system which is identical except that the leaf nodes associated with some of the level 1 nodes have been removed. More precisely, it is established that every path in the model of a system is *stuttering equivalent* (i.e. is equivalent up to sequences of repeated states) to a path in a model for a smaller system of this form. Smaller systems are produced in an inductive fashion until a star topology is reached.

We extend this proof to the parameterised family of probabilistic models based on our inductive proof schema. The probabilistic component of our model makes the proof more complex than in the non-probabilistic case. Instead of showing stuttering equivalence between paths we must establish (a probabilistic form of) stuttering equivalence (see Definition 3.5.8) between adversaries of MDPs. In this way we show that:

Theorem 5.6.2. *Let ϕ be Property 1. Then $\mathcal{M}_{\mathcal{G}} \models \phi$ for all $\mathcal{G} \in \Gamma$.*

In order to prove Theorem 5.6.2 we first show that *Property 1*, described above, is satisfied by all models in $\mathcal{M}_\Phi$, where $\Phi \subseteq \Gamma$ is the set of all star topologies. Note that $\mathcal{M}_\Phi$ is a parameterised set of models. Therefore, in order to show that all models in $\mathcal{M}_\Phi$ satisfy *Property 1*, we have to tackle another instance of the PMCP. However, the topology for this parameterised family is regular, making the problem easier. We can prove Lemma 5.6.3 using our technique for degenerative sets of models. We outline the proof below.

Lemma 5.6.3. *For all $\mathcal{G} \in \Phi$, $\mathcal{M}_{\mathcal{G}} \models \phi$, where ϕ is Property 1.*

Proof. (Sketch) Let $\mathcal{G} = (E, V, I) \in \Phi$ and suppose that $|\mathcal{G}| = N > 2$. Assume that the central process of the star has index i. For $j \neq i$, $0 \leq j \leq N-1$, let σ_N^j be a permutation on the labelling of vertices such that if $i > j$ then $\sigma_N^j(i) = 1$ and $\sigma_N^j(j) = 0$, otherwise $\sigma_N^j(i) = 0$ and $\sigma_N^j(j) = 1$. Let $W_N^j = \{i, j\} \subset V$. Define a set of reductions $Q_{\mathcal{G}} = \{(W_N^j, \sigma_N^j) | j \neq i, 0 \leq j \leq N-1\}$ on $\mathcal{G}$ and let $\mathcal{Q}_\Phi = \{Q_{\mathcal{G}} | \mathcal{G} \in \Phi, |\mathcal{G}| > 2\}$. We show that $Q_{\mathcal{G}(2)} = \emptyset$ if $\mathcal{G}(2)$ is the star of size two and Φ is reducible to $\{\mathcal{G}(2)\}$ under $\mathcal{Q}_\Phi$.

Given $\mathcal{M}_{\mathcal{G}} \in \mathbf{M}_\Phi$ with variable set $\mathbf{X}_{\mathcal{G}}$ and a reduction, $\mathcal{R}_N^j = (W_N^j, \sigma_j^N)$, for, $j \neq i$, $1 \leq j \leq N-1$, (with $i' = \sigma_j^N(i)$) define,

$$\mathbf{X}^{i'}_{\mathcal{R}_N^j(\mathcal{G})} = \mathbf{X}_{\mathcal{R}_N^j(\mathcal{G})} \setminus Y^{i'}$$

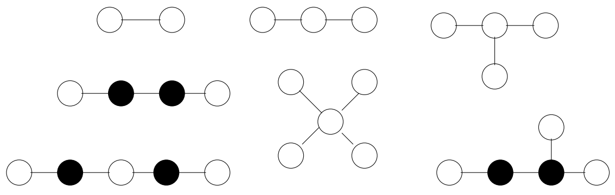


Figure 5.3: Acyclic topologies for 2, 3, 4 and 5 processes. Level-1 vertices are shaded.

where $Y^{i'} = \{child_{i',0},\ child_{i',1},\ \ldots,\ child_{i',N-1},\ adj_{i'},\ no\_of\_requests_{i'}\}$.

Suppose that process i moves into the child handshake state having received *bmp* requests from all leaf processes except some process, j. This is matched by an adversary of $\mathcal{M}_{\mathcal{G}(2)}$ in which process $\sigma^j_N(i)$ moves directly into the child handshake state.

Assume that process i moves into the child handshake state having received *bmp* requests from all leaf processes and then sends *bmc* requests to all its neighbours before receiving an *ack* from all the leaves, becoming leader. It should be clear that this is 'matched' by an adversary of $\mathcal{M}_{\mathcal{G}(2)}$ in which process $\sigma^j_N(i)$ receives a *bmp* from $\sigma^j_N(j)$ and moves into the child handshake state before sending *bmc* to $\sigma^j_N(j)$, receiving an *ack* in response and becoming leader (note that we can choose j arbitrarily in this instance).

Therefore, every adversary of $\mathcal{M}_{\mathcal{G}}$ is matched to an adversary of $\mathcal{M}_{\mathcal{G}(2)}$ and so $\mathbf{M}_\Phi$ is deterministically degenerative under $\mathcal{Q}_\Phi$ with base $\{\mathcal{G}(2)\}$. It can be shown that $\mathcal{M}_{\mathcal{G}(2)} \models \phi$, where ϕ is Property 1, using model checking (see Table 5.4) and hence by Theorem 5.5.2, $\mathcal{M}_{\mathcal{G}} \models \phi$ for all $\mathcal{G} \in \Phi$. $\square$

We can now tackle Theorem 5.6.2 by showing that $\mathcal{M}_\Gamma$ is degenerative with base Φ under some family of sets of reductions. We do so by considering each of the conditions given in Definition 5.5.1 in turn, having defined an appropriate set of complete reductions and corresponding set of 'reduced' variables for each model in $\mathcal{M}_\Gamma$.

Clipping Reductions

The main decision in defining a set of reductions is how vertices are removed from communication graphs. In the TIP example we define *clipping* reductions where we remove sets of leaf vertices that are connected to a particular kind of non-leaf vertex. These non-leaf vertices, termed *level-1* vertices, are guaranteed to exist in acyclic communication graphs that are not stars. See Figure 5.3 for examples of level-1 vertices for some acyclic graphs. The following definition of level-1 vertices is taken from [49].

Definition 5.6.4. *For any graph $\mathcal{G} = (E, V, I)$, let $leaf(v)$ denote the set of vertices, $w \in V$, such that w is a leaf and there exists an edge from v to w. A non-leaf vertex j is a level-1 vertex if all but one of its neighbouring vertices are in $leaf(j)$. If j is a level-1 vertex its non-leaf neighbour is called its inner-vertex, denoted $inner(j)$.*

Definition 5.6.6 and the proof of Lemma 5.6.5 are given in [49].

Lemma 5.6.5. *If $\mathcal{G}$ is an acyclic finite graph that is not a star, then $\mathcal{G}$ has at least two level-1 vertices.*

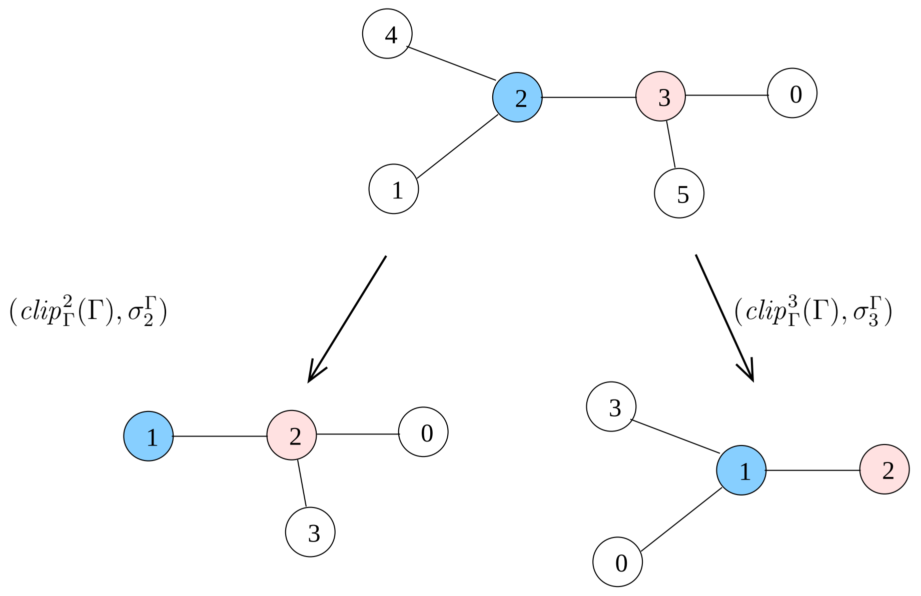
Definition 5.6.6. *Given $\mathcal{G} = (E, V, I)$ and level-1 vertex j, let $clip^j(\mathcal{G}) = V \setminus leaf(j)$ be the set of vertices of $\mathcal{G}$ excluding leaf vertices of j.*

In defining a set of complete reductions, coincident to the identification of vertices for removal, is the identification of a permutation of the vertex labels that ensures the reduced graph is labelled correctly. We define a permutation for $\mathcal{G} \in \Gamma$ and level-1 vertex j, on the vertex labels of $\mathcal{G}$, which permutes the indices such that the leaves of j have the largest indices and the order of the indices of the remaining vertices is preserved. The following definition is taken from [49]

Definition 5.6.7. *Let $\mathcal{G} \in \Gamma$ be a graph with a level-1 vertex j, with $|\mathcal{G}| = N$. Suppose that vertex j has w leaves with ids belonging to $H_j = \{h_0, h_1, \ldots, h_{w-1}\}$ where $h_u < h_{u+1}$ for all $0 \leq u \leq w-2$. The remaining $N - w$ nodes have ids in $\{0, 1, \ldots, N-1\} \setminus H_j$ which we denote by $m_0, m_1, \ldots, m_{N-1-w}$ where $m_t < m_{t+1}$ for all $0 \leq t \leq N-1-w$. We define the bijective mapping $\sigma_j^{\mathcal{G}}$ on the set $\{-1, 0, 1, \ldots, N-1\}$ such that $\sigma_j^{\mathcal{G}}(-1) = -1$, $\sigma_j^{\mathcal{G}}(m_t) = t$ for all $0 \leq t \leq N-1-w$ and $\sigma_j^{\mathcal{G}}(h_u) = u + N - w$ for all $0 \leq u \leq N - w$.*

We can now define a set of reductions on a graph $\mathcal{G} \in \Gamma$.

Definition 5.6.8. *For $\mathcal{G} \in \Gamma$, let $J_{\mathcal{G}} = \{j_1, j_2, \ldots, j_m\}$ be the set of all level-1 vertices in $\mathcal{G}$. Let $Clip_{\mathcal{G}}^j = (clip^j(\mathcal{G}), \sigma_j^{\mathcal{G}})$ and define the set of* clipping *reductions of $\mathcal{G}$ as $Clip_{\mathcal{G}} = \{Clip_{\mathcal{G}}^j | j \in J^{\mathcal{G}}\}$. Furthermore, define the family of sets of clipping reductions, $Clip_{\Gamma} = \{Clip_{\mathcal{G}} | \mathcal{G} \in |\Gamma\}$.*



4
2
3
0
1
5
$(clip^2_\Gamma(\Gamma), \sigma^\Gamma_2)$
$(clip^3_\Gamma(\Gamma), \sigma^\Gamma_3)$
1
2
0
3
3
1
2
0

Figure 5.4: An example of graph $\mathcal{G}$ (top) and the graphs obtained under a clipping reduction, with respect to level-1 vertices, vertex 2 and vertex 3.

An example of a graph obtained under clipping reductions for the level-1 vertices, 2 and 3, is shown in Figure 5.4.

We can now show that we can reduce any communication graph in Γ to a star topology under a sequence of clipping reductions.

Lemma 5.6.9. *The set of acyclic topologies, Γ is reducible to the set of stars, Φ under $Clip_\Gamma$.*

Proof. Let $\mathcal{G} \in \Phi$, then by definition $Clip^{\mathcal{G}}(\mathcal{G}) = \emptyset$.

We prove that for every $\mathcal{G} \in \Gamma \setminus \Phi$ there exists a sequence of clipping reductions such that $\mathcal{G}$ is reduced to a graph in Φ, by induction on the number of level-1 vertices of a graph. By 5.6.5 every graph that is in Γ but not a star has at least two level-1 vertices.

Assume that $\mathcal{G}$ has two level-1 vertices i and j. It should be apparent that any such graph is a line of vertices with (possibly) additional leaf vertices connected to each of the vertices in the line and with i and j at either end of the line. If the line has length two then the two vertices in the line must be the level-1 vertices. Clearly, any clipping reduction on either of these two vertices will result in a star. Suppose that every line with length $n \geq 2$ is reducible to a star by a sequence of clipping reductions. Let $\mathcal{G}$ be a line of length $n+1$.

Any clipping reduction on the level-1 vertices of $\mathcal{G}$ will result in a line of length n. Hence, by induction, any graph with two level-1 nodes is reducible to a star under a sequence of clipping reductions.

Assume that $\mathcal{G}$ has $n \geq 2$ level-1 vertices and that it is reducible to a star by a sequence of clipping reductions. Let $\mathcal{G}$ be a graph in Γ with $n+1$ level-1 vertices. Let the inner vertices of a graph $\mathcal{G}$ be the set of vertices excluding the leaf nodes and excluding the level-1 vertices. Suppose that $\mathcal{G}$ has 1 inner vertex. Then since $\mathcal{G}$ has at least three level-1 vertices, it must be a star with leaf vertices attached to the leaves of the star. Any clipping reduction on $\mathcal{G}$ will remove a level-1 vertex, with the reduced graph having n level-1 vertices and by the induction hypothesis there exists a sequence of reductions on the reduced graph resulting in a star. For any graph $\mathcal{G}$ with $n+1$ level-1 vertices and m inner vertices, assume that $\mathcal{G}$ is reducible to a star by a sequence of clipping reductions. Let $\mathcal{G}$ be a graph in Γ with $n+1$ level-1 vertices and $m+1$ inner vertices. A clipping reduction will result in either a graph with n level-1 vertices, in which case by the first induction hypothesis we are done, or in a graph with $n+1$ level-1 vertices and m inner vertices, in which case by the second induction hypothesis we are done.

Hence, by induction on the number of level-1 vertices we have that every graph in Γ is reducible to a star by a sequence of clipping reductions and therefore Γ is reducible to Φ under $Clip_\Gamma$. $\square$

Reduced Variable Sets

We define a subset of the variable set of a model of a clipping reduced graph (adapted from [49]). Essentially we remove all variables associated with the leaf processes of a level-1 vertex. This includes not just the variables of the leaf vertices but also a subset of the variables of the level-1 process that may refer to the indices of the leaf processes.

Definition 5.6.10. *Let $\mathcal{G} \in \Gamma$. Given $\mathcal{M}_\mathcal{G} \in \mathbf{M}_\Gamma$ with variables set $\mathbf{X}_\mathcal{G}$ and a clipping reduction, $Clip^j_\mathcal{G} = (clip^j(\mathcal{G}), \sigma^\mathcal{G}_j)$, for a level-1 vertex, j, (with $j' = \sigma^\mathcal{G}_j(j)$) define,*

$$\mathbf{X}^{j'}_{Clip^j_\mathcal{G}(\mathcal{G})} = \mathbf{X}_{Clip^j_\mathcal{G}(\mathcal{G})} \setminus Y^{j'}$$

where $Y^{j'} = \{child_{j',0},\ child_{j',1},\ \ldots,\ child_{j',N-1},\ adj_{j'},\ no\_of\_requests_{j'}\}$.

For $\mathcal{G} \in \Gamma$ we let $AP^{j'}_{Clip^j_\mathcal{G}(\mathcal{G})}$ be the set of atomic propositions over $\mathbf{X}^{j'}_{Clip^j_\mathcal{G}(\mathcal{G})}$.

Now we show how $\mathcal{M}_\Gamma$, with clipping reductions, is degenerative, by demonstrating that each of the conditions of Definition 5.5.1 is fulfilled.

Condition 1 (Reduced Variables and Actions)

Lemma 5.6.11. *Given a graph $\mathcal{G} \in \Gamma$ that is not a star, a level-1 vertex j, and a clipping reduction, $Clip^j_{\mathcal{G}} = (clip^j(\mathcal{G}), \sigma^j_{\mathcal{G}})$ then*

$$\mathbf{X}_{\mathcal{G}} \setminus C_{\mathcal{G}} = \mathbf{X}_{\sigma^j_{\mathcal{G}}(\mathcal{G})} \setminus C_{\mathcal{G}}, D(\mathbf{X}_{\mathcal{G}}) = D(\mathbf{X}_{\sigma^j_{\mathcal{G}}(\mathcal{G})}), Act_{\mathcal{G}} = Act_{\sigma^j_{\mathcal{G}}(\mathcal{G})}.$$

Lemma 5.6.12. *Given a graph $\mathcal{G} \in \Gamma$ that is not a star, a level-1 vertex j, and a clipping reduction, $Clip^j_{\mathcal{G}} = (clip^j(\mathcal{G}), \sigma^j_{\mathcal{G}})$ then*

$$\mathbf{X}_{Clip^j_{\mathcal{G}}(\mathcal{G})} \subseteq \mathbf{X}_{\sigma^j_{\mathcal{G}}(\mathcal{G})}, D(\mathbf{X}_{Clip^j_{\mathcal{G}}(\mathcal{G})}) \subseteq D(\mathbf{X}_{\sigma^j_{\mathcal{G}}(\mathcal{G})}), Act_{Clip^j_{\mathcal{G}}(\mathcal{G})} \subseteq Act_{\sigma^j_{\mathcal{G}}(\mathcal{G})}.$$

The proofs of Lemmas 5.6.11 and 5.6.12 follow from the definition of the variable sets $\mathbf{X}_{\mathcal{G}}$ and $Act_{\mathcal{G}}$ (for $\mathcal{G} \in \Gamma$) and the definition of $\sigma^j_{\mathcal{G}}$ given previously.

Condition 2 (Matching Adversaries)

We partition the adversaries of a TIP model according to their behaviour with respect to the level-1 vertices. Specifically, we classify them according to which level-1 vertex receives *bmp* requests from all its leaf vertices, but not its inner vertex, first. If j is such a vertex then the value of variable $state_j$ will change from *start* to *child*. The leaf neighbours of j are then guaranteed to terminate without being elected leader and their effect under the adversary can be ignored. This is key to showing that $\mathcal{M}_\Gamma$ is degenerative. The following definitions and lemmas are adapted from those given in [49] for the non-probabilistic case.

Definition 5.6.13. *Let $\mathcal{G} \in \Gamma$ and let $J^{\mathcal{G}}$ be the set of all level-1 vertices of $\mathcal{G}$. An adversary $A \in Adv_{\mathcal{M}_{\mathcal{G}}}$ is said to be* first-full *with respect to a level-1 vertex, $j \in J^{\mathcal{G}}$ (or first-full with respect to j) with inner vertex h if and only if,*

$$Prob^A_{s_0}(\{\omega \in Path^A_{s_0} | \omega \models START\, \mathcal{U}\, CHILD^h_j\}) = 1$$

where $START = (\wedge_{k \in J^{\mathcal{G}}} state_k = start)$ and, $CHILD^h_j = (\wedge_{i \in J^{\mathcal{G}}, i \neq j} state_i = start) \wedge (state_j = child) \wedge (remaining\_partner_j = h)$.

Definition 5.6.14. *Given $\mathcal{M}_{\mathcal{G}}$, for any level-1 vertex j, let $Adv^j_{\mathcal{M}_{\mathcal{G}}} \subseteq Adv_{\mathcal{M}_{\mathcal{G}}}$ denote the set of adversaries which are first-full with respect to j.*

We can now show that the set of adversaries that are first-full with respect to some level-1 vertex are the set of all adversaries for each model in $\mathcal{M}_\Gamma$. Intuitively, at the initialisation of the protocol only leaf processes can progress beyond their starting state. Thus we must reach a state where a level-1 process receives be my parent requests from all of its leaf neighbours but not its inner vertex. The adversary corresponding to this scheduling must therefore be first-full with respect to a level-1 process. The proof of Lemma 5.6.15 is similar to the proof given in [49] for the non-probabilistic case.

Lemma 5.6.15. *For $\mathcal{G} \in \Gamma \setminus \Phi$, let $J^{\mathcal{G}} = \{j_1, j_2, \ldots, j_k\}$ be the set of indices of all the level-1 vertices. Then,*

$$\bigcup_{j \in J} Adv^j_{\mathcal{M}_{\mathcal{G}}} = Adv_{\mathcal{M}_{\mathcal{G}}}.$$

Let $\mathcal{G} \in \Gamma \setminus \Phi$ and for some level-1 vertex j let $(clip^j(\mathcal{G}), \sigma^{\mathcal{G}}_j)$ be a clipping reduction of $\mathcal{G}$. Let $AP_{\mathcal{G}}$ be the set of atomic propositions over $\mathbf{X}_{\mathcal{G}}$ and let $AP_{\sigma^{\mathcal{G}}_j(\mathcal{G})}$ be the set of atomic propositions over $\mathbf{X}_{\sigma^{\mathcal{G}}_j(\mathcal{G})}$.

Let $\varsigma^{\mathcal{G}}_j$ denote the index map induced by $\sigma^{\mathcal{G}}_j$ and let $\Sigma^{\mathcal{G}}_j : AP_{\mathcal{G}} \to AP_{\sigma^{\mathcal{G}}_j(\mathcal{G})}$ denote the proposition index map induced by $\sigma^{\mathcal{G}}_j$. Note that the mapping, $\Sigma^{\mathcal{G}}_j$, is a bijective function between the labelling of states in A_j and the labelling of states in $A_{j'}$ since $\sigma^j_{\mathcal{G}}$ is a permutation. We show the effect of this mapping in Table 5.5.

Lemma 5.6.16. *Let $\mathcal{G} \in \Gamma \setminus \Phi$. Let j be a level-1 vertex and let $Clip^j_{\mathcal{G}} = (clip^j(\mathcal{G}), \sigma^j_{\mathcal{G}})$ be the clipping reduction for j. For every adversary A_j of $\mathcal{M}_{\mathcal{G}}$ that is first-full w.r.t. j, there exists some adversary $A_{j'}$ of $M_{\sigma^j_{\mathcal{G}}(\mathcal{G})}$ that is first-full with respect to $j' = \sigma^{\mathcal{G}}_j(j)$ such that $\rho^{\mathcal{G}}_j$, the path index map induced by $\sigma^j_{\mathcal{G}}$, is an isomorphism between A_j and $A_{j'}$.*

Proof. Let $\mathcal{G} \in \Gamma \setminus \Phi$ and let j be a level-1 vertex. Let $\mathcal{M}_{\mathcal{G}} = (S, s_0, Steps, Act, L)$ and $\mathcal{M}_{\sigma^j_{\mathcal{G}}(\mathcal{G})} = (S', s'_0, Steps', Act', L')$. Let A_j be an adversary of $\mathcal{M}_{\mathcal{G}}$ that is first-full with respect to j. Let α be a finite path in $Path_{fin}(s_0)$ and suppose $A_j(\alpha) = (a, \mu)$ for $(a, \mu) \in Steps(last(\alpha))$. Let $\alpha' = \rho^{\mathcal{G}}_j(\alpha)$ be the finite path in $Path_{fin}(s'_0)$ obtained under the path index map induced by $\sigma^j_{\mathcal{G}}$.

Suppose that $(a, \mu) = (r^j_m, \mu)$ corresponds to the transition type shown in Figure 5.2 with $\mu(t_1) = \mu(t_2) = 0.25$ for $t_1, t_2 \in S$ and $\mu(last(\alpha)) = 0.5$. If r^j_m is enabled from $last(\alpha)$ then since $last(\alpha') = \varsigma^{\mathcal{G}}_j(last(\alpha))$, $r^{\sigma^j_{\mathcal{G}}(j)}_{\sigma^j_{\mathcal{G}}(m)}$ is enabled from $last(\alpha')$. Let $(a', \mu') = (r^{\sigma^j_{\mathcal{G}}(j)}_{\sigma^j_{\mathcal{G}}(m)}, \mu')$ where $\mu'(last(\alpha')) = 0.5$ and $\mu'(t'_1) = \mu'(t'_2) = 0.25$ such that $t'_1, t'_2 \in S'$. It should be clear that $t'_1 = \varsigma^{\mathcal{G}}_j(t_1)$, $t'_2 = \varsigma^{\mathcal{G}}_j(t_2)$ and $\mu(t_1) = \mu(t'_1) = \mu(t_2) = \mu(t'_2) = 0.25$ and $\mu(last(\alpha)) = \mu'(last(\alpha')) = 0.5$. We can choose adversary $A_{j'}$ such that $A_{j'}(last(\alpha')) = (a', \mu')$.

We can consider each of the transitions in Tables 5.1, 5.2 and 5.3 in a similar way. Thus we can construct $A_{j'}$ so that, from Definition 3.5.18, $A_{j'}$ is isomorphic to A_j under $\rho^{\mathcal{G}}_j$.

We now prove that $A_{j'}$ is first-full with respect to $j' = \sigma^j_{\mathcal{G}}(j)$.

Consider the path,

$$\alpha = s_0 \overset{a_1,\mu_1}{\rightarrow} s_1 \overset{a_2,\mu_2}{\rightarrow} \dots \overset{a_n,\mu_n}{\rightarrow} s_n$$

such that $\alpha \in Path^{A_j}_{fin}(s_0)$, $s_n \models CHILD^h_j$ and for all $0 \leq i \leq n-1$, $s_i \models START$ and $s_i \not\models CHILD^h_j$ where h is the inner vertex of j and $CHILD^h_j$ is defined as per Definition 5.6.13. Since A_j is first-full with respect to j, α is guaranteed to exist. Root contention cannot occur before state s_n and therefore every transition along α is a non-probabilistic action and thus $\mathbf{P}(\alpha) = 1$.

Let $\alpha' = \rho^{\mathcal{G}}_j(\alpha) \in Path^{A_{j'}}(s'_0)$. Thus $\mathbf{P}(\alpha') = 1$ and for all $0 \leq i \leq n-1$,

$$\varsigma^{\mathcal{G}}_j(s_i) \models \Sigma^{\mathcal{G}}_j(START) = START,$$

$$\varsigma^{\mathcal{G}}_j(s_i) \not\models \Sigma^{\mathcal{G}}_j(CHILD^h_j) = CHILD^{h'}_{j'},$$

and

$$\varsigma^{\mathcal{G}}_j(s_i) \not\models \Sigma^{\mathcal{G}}_j(CHILD^h_j) = CHILD^{h'}_{j'},$$

where h' is the inner vertex of j'.

Thus, by Definition 5.6.13, $A_{j'}$ is first-full with respect to j'. □

In Lemma 5.6.17 we show that, for every adversary in $\mathcal{M}_{\sigma^{\mathcal{G}}_j(\mathcal{G})}$, first full with respect to $j' = \sigma^{\mathcal{G}}_j(j)$, there exists an adversary of $\mathcal{M}_{Clip^j_{\mathcal{G}}(\mathcal{G})}$, that is stuttering equivalent with respect to the set of atomic propositions defined in Definition 5.6.10.

Lemma 5.6.17. *Let $\mathcal{G} \in \Gamma \setminus \Phi$, j a level-1 vertex and $Clip^j_{\mathcal{G}} = (clip^j(\mathcal{G}), \sigma^j_{\mathcal{G}})$ be the clipping reduction for j. For every adversary A of $\mathcal{M}_{\sigma^{\mathcal{G}}_j(\mathcal{G})}$, there exists some adversary A' of $\mathcal{M}_{Clip^j_{\mathcal{G}}(\mathcal{G})}$ such that A and A' are stuttering equivalent w.r.t. $AP^{j'}_{Clip^j_{\mathcal{G}}(\mathcal{G})}$.*

Proof. Let $\mathcal{M}_{\sigma^{\mathcal{G}}_j(\mathcal{G})} = (S, s_0, Steps, Act, L)$ and $\mathcal{M}_{Clip^j_{\mathcal{G}}(\mathcal{G})} = (S', s'_0, Steps', Act', L')$. Let A be an adversary of $\mathcal{M}_{\sigma^{\mathcal{G}}_j(\mathcal{G})}$ and let $AP* \triangleq AP^{j'}_{\sigma^{\mathcal{G}}_j(\mathcal{G})}[clip^j(\mathcal{G})]$. Let $H \subseteq Path_{fin}(s_0) \times Path_{fin}(s'_0)$ be the relation given by $\forall \alpha \in Path_{fin}(s_0)$, $\alpha' \in Path_{fin}(s'_0)$, $H(\alpha, \alpha')$ iff $trace^{AP*}(\alpha) \simeq trace^{AP*}(\alpha')$.

We define an adversary, A' over $\mathcal{M}_{Clip^j_{\mathcal{G}}(\mathcal{G})}$ and sets, D_0, D_1, D_2,... such that $\forall n \geq 0$, $D_n \subseteq Path_{fin}(s'_0)$, by induction over the cuts of A at depth i. We show that for all $n \geq 0$,

IH1 For every $\alpha \in cut^A_{s_0}$, $\alpha' \in D_n$, if $H(\alpha, \alpha')$ then for every $m < n$ there exists prefixes $\beta \leq \alpha$ and $\beta' \leq \alpha'$ such that $\beta \in cut^A(m)$, $\beta' \in D_m$ and $H(\beta, \beta')$.

IH2 If μ_n, μ'_n are the distributions over $cut^n(A)$ and D_n, respectively, defined by, for $\alpha \in cut^A(n)$, $\alpha' \in D_n$, $\mu_n(\alpha) = \mathbf{P}(\alpha)$ and $\mu'_n(\alpha') = \mathbf{P}(\alpha')$ then $\mu_n \sqsubseteq_H \mu'_n$.

IH3 For every $\alpha \in cut^A_{s_0}$, $\alpha' \in D_n$, if $H(\alpha, \alpha')$ then for every $\beta \in cut^A_{s_0}$, $\beta' \in D_n$ such that $\beta \neq \alpha$ and $\beta' \neq \alpha'$, $(\beta, \alpha') \notin H$ and $(\alpha, \beta') \notin H$.

Base case: Clearly $cut^A(0) = \{s_0\}$. Let $D_0 = \{s'_0\}$. Immediately, **IH1** and **IH3** hold. By definition $\mathbf{P}(s_0) = \mathbf{P}(s'_0) = 1$. Therefore, $\mu_0 \sqsubseteq_H \mu'_0$ and so **IH2** holds.

Induction step: Assume that **IH1**, **IH2** and **IH3** hold for some $n \geq 0$. Suppose $\alpha \in cut^A(n+1)$. Then for $\gamma \in Path^A_{fin}(s_0)$, $(a, \mu) \in Steps(last(\gamma))$, $\alpha = \gamma \xrightarrow{a,\mu} s$. Since $|\gamma| = n$, $\gamma \in cut^A(n)$ and since $\mu_n \sqsubseteq_H \mu'_n$ by **IH2**, there exists $\gamma' \in D_n$ such that $H(\gamma, \gamma')$ and by **IH3** no other path is related to γ' or γ. We now define D_{n+1} by considering the transition $last(\gamma) \xrightarrow{a,\mu} s$. Consider the following cases.

1. Suppose $a \in \cup_{i \in leaf(j)} Act_i$.

   Notice that for process j to send a *be_my_parent* request to one of its leaves (k say) then it must have received a *be_my_parent* request from its inner node and all of its other leaves. This would imply, however, that A is not first-full with respect to j. Therefore, process j cannot send a *be_my_parent* request to any of its leaves and so none of the leaves can reach a contention state with j. In other words,

   $$Prob^A_{s_0}(\omega \in Path^A_{s_0} | \omega \models \Box \wedge_{i \in leaf(j)} state_i \neq contention) = 1.$$

   Thus, we only need to consider the actions $\cup_{i \in leaf(j)} \{a^i_j, b^i_j, c^i_j, d^i_j, e^i_j, f^i_j\}$ as shown in Tables 5.2 and 5.3. It is apparent that each of these actions is a non-probabilistic stutter action w.r.t. $AP*$ i.e. $\mu(s) = 1$ and $L(last(\gamma)) \cap AP* = L(s) \cap AP*$. Thus, since we also have that γ is stuttering equivalent to γ' w.r.t. $AP*$, α and γ' are stuttering equivalent w.r.t. $AP*$.

2. Suppose that $a \notin \cup_{i \in leaf(j)} Act_i$ and that $a \notin \{h^i_m | i \notin leaf(j), (m,i) \in E\}$.

   Thus a is an action associated with a process other than the leaves of j and a is not a contention resolution action. We have $\gamma \simeq \gamma'$ w.r.t. $AP*$ so $L(last(\gamma)) \cap AP* = L'(last(\gamma')) \cap AP*$. Suppose that $a = a^i_m$ for some vertex $i \notin leaf(j)$, $i \neq j$ and for some vertex m such that $(i,m) \in E$ i.e. a is the action such that a process is in the start state and receives *be_my_parent* requests from all its neighbours except m. Thus $state_i = start \in L(last(\gamma))$ and $\forall k \in V$ s.t. $(k,i) \in E$ and $k \neq m$, $c_{k,i} = bmp \in L(last(\gamma))$ and $c_{m,i} = empty \in L(last(\gamma))$. Each of these propositions are in $AP*$ and therefore also belong to $L'(last(\gamma'))$. Thus action a^i_m must also be enabled from $last(\gamma')$ and so $last(\gamma') \xrightarrow{a^i_m, \mu'} s'$ where $\mu'(s') = 1$ and $L(s) \cap AP* = L'(s') \cap AP*$.

   By arguing in a similar way for each of the actions given in Tables 5.1, 5.2 and 5.3, it is straightforward to determine that for an action $a \notin \cup_{i \in leaf(j)} Act_i$ and that $a \neq h^i_m$, if $(a,\mu) \in Steps(last(\gamma))$ then $(a,\mu') \in Steps(last(\gamma')$ where $\mu'(s') = 1$ for some $s' \in S'$ such that $L(s) \cap AP* = L(s') \cap AP*$. Let $\alpha' = \gamma' \xrightarrow{a,\mu'} s'$ then α is stuttering equivalent to α' w.r.t. $AP*$.

3. Suppose that $a \notin \cup_{i \in leaf(j)} Act_i$ and that $a \in \{h^i_m | i \notin leaf(j), (m,i) \in E\}$.

   We have $\gamma \simeq \gamma'$ w.r.t. $AP*$ so $L(last(\gamma)) \cap AP* = L'(last(\gamma')) \cap AP*$. Suppose that $a = h^i_m$ for some vertex $i \notin leaf(j)$, and for some vertex m such that $(i,m) \in E$ i.e. a is the action corresponding to contention resolution between i and m ($i < m$). Thus $state_i = contention, toss_m = 0 \in L(last(\gamma))$. Each of these propositions are in $AP*$ and therefore also belong to $L'(last(\gamma'))$. Thus action a^i_m must also be enabled from $last(\gamma')$ and so $(a^i_m, \mu') \in Steps(last(\gamma))$ where $\mu'(s') = \frac{1}{2}$, $\mu'(s'_1) = \frac{1}{4}$ and $\mu'(s'_2) = \frac{1}{4}$. If $s = last(\gamma)$ (so that $\mu(s) = \frac{1}{2}$)) then let $\alpha' = \gamma' \overset{a,\mu'}{\to} s'$. If $state_j = winner \in L(s)$ then let $\alpha' = \gamma' \overset{a,\mu'}{\to} s'_1$. If $state_j = loser \in L(s)$ then let $\alpha' = \gamma' \overset{a,\mu'}{\to} s'_2$.

We let D_{n+1} be the set of finite paths, $\{\alpha' | \alpha \in cut^A(n)$, and α' is derived from α as described above$\}$. By the definition of this set, **IH1**, **IH2** and **IH3** are satisfied. By Lemma 3.5.13, it follows that A is stuttering equivalent to A'. $\square$

Table 5.5: Result of applying $\Sigma_j^{\mathcal{G}}$ to an atomic proposition, a, for $0 \leq h, k \leq N$ and $0 \leq i \leq N-1$ (where σ is an abbreviation of $\sigma_j^{\mathcal{G}}$).

a	$\Sigma_j^{\mathcal{G}}(a)$
$position_h = x$	$position_{\sigma(h)} = x$
$child_{h,i} = x$	$child_{\sigma(h),i} = \sigma(x-1)+1$
$adj_h = x$	$adj_{\sigma(h)} = x$
$remaining\_partner_h = x$	$remaining\_partner_{\sigma(h)} = \sigma(x-1)+1$
$no\_of\_requests_h = x$	$no\_of\_requests_{\sigma(h)} = x$
$elected_h = x$	$elected_{\sigma(h)} = \sigma(x-1)+1$
$toss_h = x$	$toss_{\sigma(h)} = x$
$c_{h,k} = x$	$c_{\sigma(h),\sigma(k)} = x$

From Lemmas 5.6.15, 5.6.16 and 5.6.17 it follows that,

Lemma 5.6.18. *For every adversary A in $\mathcal{M}_{\mathcal{G}}$ ($\mathcal{G} \in \Gamma \backslash \Phi$) there exists a clipping reduction $Clip_{\mathcal{G}}^j = (clip^j(\mathcal{G}), \sigma_j^{\mathcal{G}})$ such that A is isomorphic to an adversary A' in $\mathcal{M}_{\sigma_j^{\mathcal{G}}(\mathcal{G})}$ with A' stuttering equivalent to some adversary A'' in $\mathcal{M}_{Clip_j^{\mathcal{G}}(\mathcal{G})}$ w.r.t. $AP^{j'}_{Clip_j^{\mathcal{G}}(\mathcal{G})}$ (for $j' = \sigma_j^{\mathcal{G}}(j)$).*

5.6.5 Proof of Theorem 5.6.2

Proof. From Lemma 5.6.9 it follows that the set of clipping reductions is reducible to Φ. By Lemmas 5.6.11, 5.6.12 and 5.6.18, the conditions of Definition 5.5.1 are satisfied by $\mathcal{M}_\Gamma$ (under the clipping reductions). Thus, $\mathcal{M}_\Gamma$ is deterministically degenerative with base Φ. By Lemma 5.6.3 *Property 1* holds for all star topologies. Since *Property 1* is unindexed and has appropriately defined propositions, by Theorem 5.5.2, it is satisfied by $\mathcal{M}_{\mathcal{G}}$ for all $\mathcal{G} \in \Gamma$. □

5.7 Extending the Properties we can prove for deterministically degenerative families of models

In the previous sections we considered QLTL properties that were unindexed. We extend the class of properties that we can prove using our inductive proof schema to indexed properties of a certain form. In particular, we define a QLTL$_{\backslash\mathcal{X}}$ property ϕ to be I-indexed if it has the form,

$$\mathcal{P}_{\geq p}[\vee_{i \in \mathcal{I}} \psi_i] \text{ or, } \mathcal{P}_{\leq p}[\wedge_{i \in \mathcal{I}} \psi_i].$$

The ψ_i are (isomorphic up to indexing) LTL path formulae indexed only by $i \in \mathcal{I}$. For example, suppose that $I = \{0, 1, 2\}$, and that x_0, x_1, x_2 are indexed variables, each with domain {0,1}. Then

$$\mathcal{P}_{\geq 0.75}[(\Diamond(x_0 = 1)) \vee (\Diamond(x_1 = 1)) \vee (\Diamond(x_2 = 1))]$$

is an I-indexed QLTL$_{\backslash\mathcal{X}}$ property.

Theorem 5.7.1. *Let $\mathcal{S}$ be a specification over a set of communication topologies, Γ, where Γ is reducible to Φ under the family of sets of reductions, $\mathcal{Q}_\Gamma$. Suppose that the set of models, $\mathbf{M}_\Gamma$, is deterministically degenerative under $\mathcal{Q}_\Gamma$ with base Φ. For $\mathcal{G} = (E, V, I) \in \Gamma$, let $I(\mathcal{G})$ denote the set of indices of $\mathcal{G}$. For every $\mathcal{F} \in \Phi$, let $\phi(I(\mathcal{F}))$ be an $I(\mathcal{F})$-indexed quantitative QLTL$_{\backslash\mathcal{X}}$ property with atomic propositions in $\bigcap_{\mathcal{G}\in\Gamma} \bigcap_{(W,\sigma)\in Q_\mathcal{G}} AP'_{\sigma(\mathcal{G})[W]}$. If $\mathcal{M}_\mathcal{F} \models \phi(I(\mathcal{F}))$ for all $\mathcal{F} \in \Phi$, $\mathcal{M}_\mathcal{G} \models \phi(I(\mathcal{G}))$ for all $\mathcal{G} \in \Gamma$.*

Proof. Let $\mathcal{S}$ be a system and Γ a set of communication topologies. Suppose that there exists a set of models,

$$\mathbf{M}_\Gamma = \{\mathcal{M}_\mathcal{G} \text{ over } \mathbf{X}_\mathcal{G} | \mathcal{G} \in \Gamma\},$$

for $\mathcal{S}$ over Γ, with sets of atomic propositions $AP_\mathcal{G}$ over $\mathbf{X}_\mathcal{G}$, that is deterministically degenerative with base Φ, under $\mathcal{Q}_\Gamma$, a family of sets of reductions such that Γ is reducible to Φ under $\mathcal{Q}_\Gamma$. Assume that $\mathcal{G} \in \Gamma$ is not in Φ and let $Q_\mathcal{G}$ be the set of reductions of $\mathcal{G}$ in $\mathcal{Q}_\Gamma$.

For each $\mathcal{G} \in \Gamma$ and $\mathcal{R} = (W, \sigma) \in Q_\mathcal{G}$, let $\mathbf{X}'_{\mathcal{R}(\mathcal{G})} \subseteq \mathbf{X}_{\mathcal{R}(\mathcal{G})}$ be a set of variables such that Condition 2 of Definition 5.5.1 holds. Let $AP'_{\mathcal{R}(\mathcal{G})} \subseteq AP_{\mathcal{R}(\mathcal{G})}$, be the set of atomic propositions over $\mathbf{X}'_{\mathcal{R}(\mathcal{G})}$.

For each $\mathcal{R} = (W, \sigma) \in Q_\mathcal{G}$, let $\phi(I(\mathcal{R}(\mathcal{G})))$ be an $I(\mathcal{R}(\mathcal{G}))$-indexed QLTL$_{\backslash\mathcal{X}}$ property with atomic propositions in $\bigcap_{\mathcal{R}\in Q_\mathcal{G}} AP'_{\mathcal{R}(\mathcal{G})}$. Suppose that $\mathcal{M}_{\mathcal{R}(\mathcal{G})} \models \phi(I(\mathcal{R}(\mathcal{G})))$, for every $\mathcal{R} \in Q^\mathcal{G}$. Then we can show that $\mathcal{M}_\mathcal{G} \models \phi(I(\mathcal{G}))$ as follows.

Let $A \in Adv_{\mathcal{M}_\mathcal{G}}$. Choose $\mathcal{R} = (W, \sigma) \in Q_\mathcal{G}$ such that A is isomorphic to some adversary $A' \in Adv_{\mathcal{M}_{\sigma(\mathcal{G})}}$ under the path index isomorphism, ρ, induced by σ, with A' stuttering equivalent to some adversary $A'' \in Adv_{\mathcal{M}_{\mathcal{R}(\mathcal{G})}}$ w.r.t. $AP'_{\mathcal{R}(\mathcal{G})}$. The property $\phi(I(\mathcal{R}(\mathcal{G})))$

has the form $\mathcal{P}_{\geq p}[\vee_{i \in I(\mathcal{R}(\mathcal{G}))}\psi_i]$ or $\mathcal{P}_{\leq p}[\bigwedge_{i \in I(\sigma(\mathcal{G})[W])}\psi_i]$. We consider only the former here (the proof of the latter case is similar). Note that,

$$\{\omega \models \vee_{i \in I(\mathcal{R}(\mathcal{G}))}\psi_{\sigma^{-1}(i)}\} \subseteq \{\omega \models \vee_{i \in I(\mathcal{G})}\psi_{\sigma^{-1}(i)}\}) \tag{5.2}$$

For every adversary B of $\mathcal{M}_{\mathcal{R}(\mathcal{G})}$,

$$Prob^B_{s_0''}(\{\omega'' \in Path^B_{s_0''} | \omega'' \models \vee_{i \in I(\mathcal{G})}\psi_i\}) \geq p. \tag{5.3}$$

If $\mathcal{M}_{\mathcal{G}}$, $\mathcal{M}_{\sigma(\mathcal{G})}$ and $\mathcal{M}_{\mathcal{R}(\mathcal{G})}$ have initial states s_0, s_0' and s_0'' respectively then,

$Prob^A_{s_0}(\{\omega \in Path^A_{s_0} | \omega \models \vee_{i \in I(\mathcal{G})}\psi_i\})$

$= Prob^A_{s_0}(\{\omega \in Path^A_{s_0} | \omega \models \vee_{i \in I(\mathcal{G})}\psi_{\sigma^{-1}(i)}\})$ since σ is bijective

$\geq Prob^A_{s_0}(\{\omega \in Path^A_{s_0} | \omega \models \vee_{i \in I(\mathcal{R}(\mathcal{G}))}\psi_{\sigma^{-1}(i)}\})$ by 5.2

$= Prob^{A'}_{s_0'}(\{\omega' \in Path^{A'}_{s_0'} | \omega' \models \Sigma(\vee_{i \in I(\mathcal{R}(\mathcal{G}))}\psi_{\sigma^{-1}(i)})\})$ since $A = A'$ under ρ

$= Prob^{A'}_{s_0'}(\{\omega' \in Path^{A'}_{s_0'} | \omega' \models \vee_{i \in I(\mathcal{R}(\mathcal{G}))}\psi_i\})$ since Σ is induced by σ

$= Prob^{A''}_{s_0''}(\{\omega'' \in Path^{A''}_{s_0''} | \omega'' \models \vee_{i \in I(\mathcal{R}(\mathcal{G}))}\psi_i\})$ since $A \simeq A'$ w.r.t. $AP'_{\mathcal{R}(\mathcal{G})}$

$\geq p$ by 5.3.

Since the above is true for every adversary of $\mathcal{M}_{\mathcal{G}}$, $\mathcal{M}_{\mathcal{G}} \models \phi(I(\mathcal{G}))$.

For each $\mathcal{F} \in \Phi$ let $\phi(I(\mathcal{F}))$ be an $I(\mathcal{F})$-indexed quantitative LTL$_{\backslash\mathcal{X}}$ formula with atomic propositions in $\bigcap_{\mathcal{G} \in \Gamma} \bigcap_{\mathcal{R} \in Q_{\mathcal{G}}} AP'_{\mathcal{R}(\mathcal{G})}$ and let $\mathcal{G} \in \Gamma$. If $\mathcal{G} \in \Phi$ then, by the statement of the theorem, $\mathcal{M}_{\mathcal{G}} \models \phi(I(\mathcal{G}))$. Otherwise, $\mathcal{G} \in \Gamma \setminus \Phi$ and since $\phi(I(\mathcal{G}))$ is $I(\mathcal{G})$-indexed, with atomic propositions in $\bigcap_{\mathcal{G} \in \Gamma} \bigcap_{\mathcal{R} \in Q_{\mathcal{G}}} AP'_{\mathcal{R}(\mathcal{G})}$, $\phi(I(\mathcal{G}))$ is defined over $\cap_{\mathcal{R} \in Q_{\mathcal{G}}} AP'_{\mathcal{R}(\mathcal{G})}$. Hence, by the above, $\mathcal{M}_{\mathcal{G}} \models \phi(I(\mathcal{G}))$ if $\mathcal{M}_{\sigma(\mathcal{G})[W]} \models \phi(I(\sigma(\mathcal{G}))[W])$ for all $(W, \sigma) \in Q_{\mathcal{G}}$. For each $\mathcal{R} \in Q_{\mathcal{G}}$, either $\mathcal{R}(\mathcal{G})$ is in Φ or it can be reduced further. Continuing in this way, since Γ is reducible to Φ under $\mathcal{Q}_{\Gamma}$, we can construct a tree of graphs in which every terminal node is a graph in Φ. Finally, by statement of the theorem, each model associated with the graph, $\mathcal{F}$ at these terminal nodes satisfy $\psi(I(\mathcal{F}))$ and, by propagation up the tree of graphs, it follows that $\mathcal{M}_{\mathcal{G}} \models \phi(I(\mathcal{G}))$. $\square$

5.8 Discussion

5.8.1 Analysing the TIP

Much work has been carried out on analysing the TIP (see for example, [46]). We mention [3] since it describes an inductive proof for a spanning tree leader election protocol that is similar to the TIP. The authors observe that only a leaf can initially transmit an "up" (*bmp*) message and it will then move to a "dead" state, after which the protocol behaves as if started in the graph with that leaf deleted. They note that, continuing in this manner, eventually a graph with only one or two vertices will be reached. The protocol is not specified formally and, unlike our approach, is not verified using state-based methods.

Summary We have described an inductive proof technique for a class of randomised distributed systems described as *degenerative.* The systems are modelled as MDPs. The technique is an induction schema over the underlying communication topologies, represented by undirected graphs. The key idea is that topologies are *reduced* such that every adversary of a model of a system with some topology is stuttering equivalent to an adversary of a model of a system with a reduced topology. Reduction involves the removal of one or more vertices from the communication topology. The base case(s) are those topologies that are not reduced. We have applied this technique to the IEEE 1394 (Firewire) tree identify protocol showing that certain unindexed $\mathrm{QLTL}_{\backslash\mathcal{X}}$ properties that are true of the system with a base topology will hold for a system with *any* acyclic topology. In this case, reduction is by removal of the leaf vertices of level one vertices. The base cases are star topologies.

Chapter 6

Probabilistic Parameterised Verification of Probabilistically Degenerative Systems

Outline In this chapter we extend the definition of a deterministically degenerative family of MDP models of a randomised distributed system. We define a probabilistically degenerative family of models, and show that a family of MDP models of the Itai Rodeh leader election protocol for rings satisfy this definition. Establishing that a family of models is probabilistically degenerative enables us to verify certain properties of the family by considering just a subset of models. We use this result to determine properties of the Itai Rodeh leader election protocol.

6.1 Introduction

We note that our technique described in the previous chapter is only applicable to families of models of systems under which processes degenerate *deterministically*. For example, in the TIP case study, a level-1 process is determined to be degenerate according to the scheduling of the leaves of all the level-1 processes. Since the scheduling of processes is modelled non-deterministically, under any adversary it is known which level-1 process degenerates. Therefore the protocol degenerates deterministically and so our technique is applicable in this instance.

In the TIP, the probabilistic element of the protocol is restricted to the resolution of contention and is not relevant to the removal of the leaves of a level-1 process. However, for other randomised distributed protocols that can be deemed degenerative, processes are removed in a probabilistic fashion. Consider, for example, the Itai-Rodeh leader election protocol for rings. Initially all processes are active and, in each round of the protocol, a subset of the processes is chosen to become passive according to a simple probabilistic choice made by each process. Since the passive processes only transmit received messages they can be considered to be degenerate processes. Since the passive processes are chosen randomly, the protocol can be considered *probabilistically* degenerative.

We therefore extend our definition of a degenerative family of models to include those that degenerate probabilistically. The main difference from our original definition is that we consider a set of *distributions* over graph reductions, such that every adversary of a model of a system with some communication graph is stuttering equivalent to a set of paths of an adversary of a model of a system with a reduced communication graph.

Note that in the case of deterministically degenerative families of models we did not consider infinite cyclic behaviour i.e. where we continuously return to a state that we previously visited having performed some set of actions. For a model to degenerate deterministically, it cannot exhibit cyclic behaviour (unless under some set of fairness constraints). In the probabilistic case, however, a model can cycle and still be degenerative, assuming that the cycle is probabilistic i.e. occurs with probability less than one. Therefore, the point of execution when the system degenerates is probabilistic. In other words the system degenerates at different times. In terms of a model of a system this represents the case that for sets of paths under some adversary there are different states when the system degenerates.

6.1.1 Incomplete Cyclic Reductions

It should be apparent that our earlier definition of reductions did not take into account families of graphs that are cyclic, such as rings. To remedy this we extend our definition to allow reductions to include an insertion operation that can add edges to a subgraph. In the sequel, for a graph $\mathcal{G} = (E, V, I)$ and a set of edges $E' \subseteq \bar{E}$ (where $\bar{E}$ is the set of edges in the complement of $\mathcal{G}$), let $\mathcal{G} \oplus E' = (E \cup E', V, I)$.

Definition 6.1.1. *Let Γ be a set of graphs and let $\mathcal{G} = (E, V, I) \in \Gamma$. Let σ be a permutation of vertex labels, let $V' \subset V$ and let E' be a subset of the set of edges in the complement of $\mathcal{G}[V']$. The tuple (E', V', σ) is a* cyclic reduction *of $\mathcal{G}$ in Γ if and only if $\sigma(\mathcal{G})[V'] \oplus E' \in \Gamma$. We describe $\sigma(\mathcal{G})[V'] \oplus E'$ as a (the) cyclically reduced graph of $\mathcal{G}$ (with E' under V' by σ).*

When considering cyclic behaviours, as discussed above, in probabilistic degenerative systems we must consider behaviours such that no set of processes degenerate. In terms of the communication topology of a system we therefore must further extend our definition of graph reductions to include an *identity* reduction under which a graph is 'reduced' to itself.

Definition 6.1.2. *Let Γ be a set of graphs and let $\mathcal{G} = (E, V, I) \in \Gamma$. Let σ be a permutation of vertex labels, $V' \subseteq V$ and $F \subseteq \bar{E}'$ (for $\mathcal{G}[V'] = (E', V', I')$). Suppose that either $V' = V$, $\sigma = \iota$ (the identity permutation) and $F = \emptyset$ or $V' \subset V$. The tuple, $\mathcal{R} = (F, V', \sigma)$ is then an incomplete cyclic reduction of $\mathcal{G}$ in Γ if and only if $\mathcal{R}(\mathcal{G}) = \sigma(\mathcal{G})[V'] \oplus F \in \Gamma$. The reduction $(\emptyset, V, \iota)$ is the identity reduction on $\mathcal{G}$.*

We give an extension to our definition of a reducible family of graphs (see Definition 5.2.6). Here we consider a family of *distributions* over *incomplete cyclic* reductions. In the sequel let $\mathit{IRed}_{\mathcal{G}}$ denote the set of all incomplete cyclic reductions of $\mathcal{G}$.

Definition 6.1.3. *Let Γ be a topology and let $\mathcal{Q}_\Gamma = \{\mu_{\mathcal{G}} | \mathcal{G} \in \Gamma\}$ be a family of distributions over incomplete cyclic reductions for graphs in Γ where, for $\mathcal{G} \in \Gamma$, $\mu_{\mathcal{G}} : \mathit{IRed}_{\mathcal{G}} \to [0, 1]$ is a discrete distribution over the set of reductions of $\mathcal{G}$. Let $\Phi \subset \Gamma$ such that for all $\mathcal{G} = (E, V, I) \in \Phi$, $\mu_{\mathcal{G}}((\emptyset, V, \iota)) = 1$.*

It is possible to construct an infinite state DTMC $\mathcal{G}_\Gamma = (\Gamma, \Gamma, \mathbf{P}, L)$ over the set of propositions $AP = \{g = 0, g = 1\}$ where, for $\mathcal{G}, \mathcal{G}' \in \Gamma$,

$$\mathbf{P}(\mathcal{G}, \mathcal{G}') = \sum_{\mathcal{R} \in support(\mu_{\mathcal{G}}), \mathcal{G}' = \mathcal{R}(\mathcal{G})} \mu_{\mathcal{G}}(\mathcal{R})$$

with, for all $\mathcal{G} \in \Gamma$, $L(\mathcal{G}) = \{g = 1\}$ if $\mathcal{G} \in \Phi$, $L(\mathcal{G}) = \{g = 0\}$, otherwise. Then Γ is reducible to Φ with probability 1 under $\mathcal{Q}_\Gamma$ if and only if, for all $\mathcal{G} \in \Gamma$, $\mathcal{G} \models \mathcal{P}_{\geq 1}[\Diamond(g = 1)]$.

6.1.2 Probabilistically Degenerative Families of Models

Whereas for deterministically degenerative sets of models we know which processes will degenerate under each adversary, for probabilistically degenerative sets of models under an adversary different sets of processes will degenerate with associated probabilities. We therefore need to *partition* the sets of paths under an adversary.

Definition 6.1.4. *Let $\mathcal{M}$ be a MDP and A be an adversary of $\mathcal{M}$. A set $\{P_0, P_1, \ldots, P_n\}$ is a measurable partition of $Path^A_{s_0}$ if and only if $\forall 0 \leq i \leq n$, $P_i \subseteq Path^A_{s_0}$ such that P_i is a measurable set of paths with, for $0 \leq j \leq n$, $i \neq j$ $P_i \cap P_j = \emptyset$ and $\cup^n_{i=0} P_i = Path^A(s_0)$.*

Proposition 6.1.5 follows from the definition of a measurable partition and standard results of probability theory (namely the Law of Total Probability).

Proposition 6.1.5. *Let $\mathcal{M}$ be a MDP and let A be an adversary of $\mathcal{M}$. For a measurable partition, $\{P_0, \ldots, P_n\}$ of $Path^A_{s_0}$ and a LTL path formula, ψ,*

$$Prob^A_{s_0}(\{\omega \in Path^A_{s_0} | \omega \models \psi\}) = \sum_{i=0}^{n} Prob^A_{s_0}(\{\omega \in P_i | \omega \models \psi\}).$$

We now give a definition of a probabilistically degenerative set of models.

Definition 6.1.6. *Let Γ be a communication topology that is reducible to some topology Φ under a family of distributions over sets of incomplete cyclic reductions, $\mathcal{Q}_\Gamma = \{\mu_\mathcal{G} | \mathcal{G} \in \Gamma\}$. Furthermore, suppose $\mathcal{S}(\Gamma)$ is set of specifications over Γ, and let $\mathcal{M}_\Gamma = \{\mathcal{M}_\mathcal{G} | \mathcal{G} \in \Gamma\}$ be the set of models over $\mathcal{S}(\Gamma)$.*

For $\mathcal{G} \in \Gamma$ let $X_\mathcal{G}$ be the set of variables of $\mathcal{S}(\mathcal{G})$ (with $C_\mathcal{G} \subseteq X_\mathcal{G}$ the set of channel variables) and let $AP_\mathcal{G}$ the set of atomic propositions over $X_\mathcal{G}$. For each $\mathcal{G} \in \Gamma$ and each $\mathcal{R} \in support(\mu_\mathcal{G})$ that is not the identity reduction, let $X'_{\mathcal{R}(\mathcal{G})} \subseteq \mathbf{X}_{\mathcal{R}(\mathcal{G})}$ be a set of variables (with $AP'_{\mathcal{R}(\mathcal{G})} \subseteq AP_{\mathcal{R}(\mathcal{G})}$, the set of atomic propositions over $X'_{\mathcal{R}(\mathcal{G})}$). Then $\mathcal{M}_\Gamma$ is probabilistically degenerative with base Φ under $\mathcal{Q}_\Gamma$ if and only if,

1. *(**Reduced Variables and Actions:**)For $\mathcal{G} \in \Gamma$ and a complete cyclic reduction $\mathcal{R} = (F, R, \sigma)$ where $\mu_\mathcal{G}(\mathcal{R}) > 0$,*

$$X_{\sigma(\mathcal{G})} \setminus C_\mathcal{G} = X_\mathcal{G} \setminus C_\mathcal{G}, D(X_{\sigma(\mathcal{G})}) = D(X_\mathcal{G}), Act_{\sigma(\mathcal{G})} = Act_\mathcal{G},$$

$$X_{\mathcal{R}(\mathcal{G})} \subseteq X_{\sigma(\mathcal{G})}, D(X_{\mathcal{R}(\mathcal{G})}) \subseteq D(X_{\sigma(\mathcal{G})}), Act_{\mathcal{R}(\mathcal{G})} \subseteq Act_{\sigma(\mathcal{G})},$$

2. ***(Matching Adversaries:)*** *For $\mathcal{G} \in \Gamma \setminus \Phi$, for every deterministic adversary A of $\mathcal{M}_{\mathcal{G}}$, if*

$$support(\mu_{\mathcal{G}}) = \{\mathcal{R}_0 = (F_0, R_0, \sigma_0), \mathcal{R}_1 = (F_1, R_1, \sigma_1), \ldots, \mathcal{R}_m = (F_m, R_m, \sigma_m)\},$$

there exists a measurable partition $P = \{P_0, P_1, \ldots, P_m\}$ of $Path^A_{s_0}$ with (for $0 \leq j \leq m$), $Prob^A_{s_0}(\{\alpha \in P_j\}) = \mu_{\mathcal{G}}(\mathcal{R}_j)$ and P_j isomorphic to P'_j under the path index map induced by σ for some $P'_j \subseteq Path^{A'}_{s'_0}$ for some adversary A' of $\mathcal{M}_{\sigma_j(\mathcal{G})}$ such that, P'_j is stuttering equivalent, with respect to $AP'_{\mathcal{R}(\mathcal{G})}$, to E'', for some randomised adversary E'' of $\mathcal{M}_{\mathcal{R}(\mathcal{G})}$.

We then have that,

Theorem 6.1.7. *Let Γ be a communication topology that is reducible to Φ under the family of distributions over sets of incomplete cyclic reductions, $\mathcal{Q}_\Gamma$ and let $\mathcal{S}(\Gamma)$ be a parameterised specification over Γ. Suppose that for each $\mathcal{G} \in \Gamma$, $\mathcal{R} \in support(\mu_{\mathcal{G}})$, there is a set of variables $X'_{\mathcal{R}(\mathcal{G})} \subseteq X_{\mathcal{R}(\mathcal{G})}$ (with $AP'_{\mathcal{R}(\mathcal{G})} \subseteq AP_{\mathcal{R}(\mathcal{G})}$, the set of atomic propositions over $X'_{\mathcal{R}(\mathcal{G})}$) such that the set of models over $\mathcal{S}(\Gamma)$, $\mathbf{M}_\Gamma$, is probabilistically degenerative with base Φ under $\mathcal{Q}_\Gamma$. Then for any unindexed QLTL$_{\setminus\mathcal{X}}$ property ϕ with atomic propositions in $\bigcap_{\mathcal{G}\in\Gamma} \bigcap_{\mathcal{R}\in support(\mu_{\mathcal{G}})} AP'_{\mathcal{R}(\mathcal{G})}$, if $\mathcal{M}_{\mathcal{F}} \models \phi$ for all $\mathcal{F} \in \Phi$, $\mathcal{M}_{\mathcal{G}} \models \phi$ for all $\mathcal{G} \in \Gamma$.*

Proof. Assume that $\mathcal{G} \in \Gamma$ is not in Φ, and let $\mu_{\mathcal{G}}$ be the distribution defined over reductions for $\mathcal{G}$. Let ϕ be an unindexed QLTL$_{\setminus\mathcal{X}}$ property with atomic propositions in $\mathbf{AP} = \bigcap_{\mathcal{R}\in support(\mu_{\mathcal{G}})} AP'_{\mathcal{R}(\mathcal{G})}$. Suppose that $\mathcal{M}_{\mathcal{R}(\mathcal{G})} \models \phi$, for every *complete* reduction, $\mathcal{R} = (F, R, \sigma) \in support(\mu_{\mathcal{G}})$ (i.e. $\mathcal{R} \neq (\emptyset, V, \iota)$). We show that $\mathcal{M}_{\mathcal{G}} \models \phi$ as follows.

Let Σ be the proposition index map induced by σ. Note that ϕ has the form $\mathcal{P}_{\bowtie p}[\psi]$ and that $\Sigma(\psi) = \psi$ since ϕ is unindexed and that, for every adversary B of $\mathcal{M}_{\mathcal{R}(\mathcal{G})}$,

$$Prob^B_{s''_0}(\{\omega \in Path^B_{s''_0} | \omega \models \psi\}) \bowtie p. \tag{6.1}$$

Suppose that

$$support(\mu_{\mathcal{G}}) = \{\mathcal{R}_0 = (F_0, R_0, \sigma_0), \mathcal{R}_1 = (F_1, R_1, \sigma_1), \ldots, \mathcal{R}_m = (F_m, R_m, \sigma_m)\}$$

and that $P = \{P_0, P_1, \ldots, P_m\}$ is a partition of $Path^A_{s_0}$ with (for $0 \leq j \leq m$),

$$Prob^A_{s_0}(\{\alpha \in P_j\}) = \mu_{\mathcal{G}}(\mathcal{R}_j)$$

and P_j isomorphic to P'_j under the path index isomorphism induced by σ, ρ, say, for some $P'_j \subseteq Path^{A'}_{s'_0}$ for some adversary A'_j of $\mathcal{M}_{\sigma_j(\mathcal{G})}$ such that P'_j is stuttering equivalent, with respect to $AP'_{\mathcal{R}_j(\mathcal{G})}$, to E''_j, for some randomised adversary E''_j of $\mathcal{M}_{\mathcal{R}_j(\mathcal{G})}$. Let $\mathcal{M}_{\mathcal{G}}$, $\mathcal{M}_{\sigma_j(\mathcal{G})}$ and $\mathcal{M}_{\mathcal{R}_j(\mathcal{G})}$ have initial states s_0, $s^{j'}_0$ and $s^{j''}_0$ respectively.

We consider two separate cases. One, that every reduction of $\mathcal{G}$ is a complete reduction and, two, that one of the reductions is the identity reduction. Suppose that $(\emptyset, V, \iota) \notin$ *support*$(\mu_{\mathcal{G}})$ and let $A \in Adv_{\mathcal{M}_{\mathcal{G}}}$, then,

$$\begin{array}{rl}
& Prob^A_{s_0}(\{\omega \in Path^A_{s_0} | \omega \models \psi\}) \\
= & \sum_{i=0}^{m} Prob^A_{s_0}(\{\omega \in P_i | \omega \models \psi\}) \text{ since } P \text{ is a measurable partition} \\
= & \sum_{i=0}^{m} Prob^{A'_i}_{s^{i'}_0}(\{\omega' \in P'_i | \omega' \models \Sigma(\psi)\}) \text{ since } \psi \text{ has propositions in } \mathbf{AP} \text{ and } P'_i = P_i \\
 & \\
= & \sum_{i=0}^{m} Prob^{A'_i}_{s^{i'}_0}(\{\omega' \in P'_i | \omega' \models \psi\}) \text{ since } \psi \text{ is unindexed} \\
= & \sum_{i=0}^{m} Prob^{A'_i}_{s^{i'}_0}(P'_i).Prob^{E''_i}_{s^{i''}_0}(\{\omega'' \in Path^{E''_i}_{s^{i''}_0} | \omega'' \models \psi\}) \text{ since } P'_i \simeq E''_i \text{ w.r.t. } AP'_{\mathcal{R}(\mathcal{G})} \\
\bowtie & \sum_{i=1}^{m} Prob^{A'_i}_{s^{i'}_0}(P'_i).p \text{ by 6.1} \\
= & p.\sum_{i=0}^{m} Prob^A_{s_0}(P_i) \text{ since } P'_i = P_i \\
= & p \text{ since } P \text{ is measurable partition and so } \sum_{i=0}^{m} Prob^A_{s_0}(P_i) = 1
\end{array}$$

Since the above is true for every adversary of $\mathcal{M}_{\mathcal{G}}$, $\mathcal{M}_{\mathcal{G}} \models \phi$.

Now, suppose that $(\emptyset, V, \iota) \in$ *support*$(\mu_{\mathcal{G}})$ and, without loss of generality, that $\mathcal{R}_0 = (\emptyset, V, \iota)$. Let $G_\Gamma = (\Gamma, \Gamma, \mathbf{P}, L)$ be the infinite state DTMC constructed as per Definition 6.1.3. Also, for $\bowtie \in \{>, \geq <, \leq\}$, define $\underline{\bowtie} \in \{\geq, \leq\}$ such that, $\underline{\geq} \equiv \geq$, $\underline{>} \equiv \geq$, $\underline{\leq} \equiv \leq$, $\underline{<} \equiv \leq$.

$$\begin{array}{rl}
& Prob^{A}_{s_0}(\{\omega \in Path^{A}_{s_0} | \omega \models \psi\}) \\
= & \displaystyle\sum_{i=0}^{m} Prob^{A}_{s_0}(\{\omega \in P_i | \omega \models \psi\}) \text{ since } P \text{ is a measurable partition} \\
= & \displaystyle\sum_{i=0}^{m} Prob^{A'_i}_{s^{i'}_0}(\{\omega' \in P'_i | \omega' \models \Sigma(\psi)\}) \text{ since } P'_i = P_i \\
= & \displaystyle\sum_{i=0}^{m} Prob^{A'_i}_{s^{i'}_0}(\{\omega' \in P'_i | \omega' \models \psi\}) \text{ since } \psi \text{ is unindexed} \\
= & \displaystyle\sum_{i=0}^{m} Prob^{A'_i}_{s^{i'}_0}(P'_i).Prob^{E''_i}_{s^{i''}_0}(\{\omega'' \in Path^{E''_i}_{s^{i''}_0} | \omega'' \models \psi\}) \text{ since } P'_i \simeq E''_i \text{ w.r.t. } AP'_{\mathcal{R}_j(\mathcal{G})} \\
= & \displaystyle\sum_{i=0}^{m} Prob^{A}_{s_0}(P_i).Prob^{E''_i}_{s^{i''}_0}(\{\omega'' \in Path^{E''_i}_{s^{i''}_0} | \omega'' \models \psi\}) \text{ since } P'_i = P_i \\
\bowtie & Prob^{A}_{s_0}(P_0).Prob^{E''_0}_{s_0}(\{\omega'' \in Path^{E''_0}_{s_0} | \omega'' \models \psi\}) \\
& \displaystyle+\sum_{i=1}^{m} Prob^{A}_{s_0}(P_i).p \text{ by (6.1)} \\
\underline{\bowtie} & Prob^{A}_{s_0}(P_0).Prob^{A_0}_{s_0}(\{\omega \in Path^{A_0}_{s_0} | \omega \models \psi\}) \\
& \displaystyle+\sum_{i=1}^{m} Prob^{A}_{s_0}(P_i).p \text{ for } A_0 \in Adv_{\mathcal{M}_\mathcal{G}} \text{ by Lemma 3.3.4} \\
= & \mu_\mathcal{G}((\emptyset, V, \iota)).Prob^{A_0}_{s_0}(\{\omega \in Path^{A_0}_{s_0} | \omega \models \psi\}) \\
& \displaystyle+p.\sum_{i=1}^{m} \mu_\mathcal{G}(\mathcal{R}_i) \text{ from Definition 6.1.6}
\end{array}$$

Since A is arbitrary in the above, for some $A_1 \in Adv_{\mathcal{M}_\mathcal{G}}$,

$$\begin{array}{rcl}
Prob^{A_0}_{s_0}(\{\omega \in Path^{A_0}_{s_0} | \omega \models \psi\}) & \bowtie & \mu_\mathcal{G}((\emptyset, V, \iota)).Prob^{A_1}_{s_0}(\{\omega \in Path^{A_1}_{s_0} | \omega \models \psi\}) \\
& + & \displaystyle p.\sum_{i=1}^{m} \mu_\mathcal{G}((F_i, R_i, \sigma_i))
\end{array}$$

Substituting this into the above,

$$\begin{array}{rcl}
Prob^{A}_{s_0}(\{\omega \in Path^{A_0}_{s_0} | \omega \models \psi\}) & \bowtie & \mu_\mathcal{G}(\emptyset, V, \iota))^2.Prob^{A_1}_{s_0}(\{\omega \in Path^{A_1}_{s_0} | \omega \models \psi\}) \\
& + & \displaystyle \mu_\mathcal{G}(\emptyset, V, \iota)).p.\sum_{i=1}^{m} \mu_\mathcal{G}(\mathcal{R}_i) \\
& + & \displaystyle p.\sum_{i=1}^{m} \mu_\mathcal{G}((\mathcal{R}_i))
\end{array}$$

Continuing in this manner,

$$\begin{aligned}
\mathit{Prob}^{A}_{s_0} &\bowtie \sum_{j=1}^{\infty} \mu_{\mathcal{G}}(\emptyset, V, \iota)^{j-1}.p.\sum_{i=1}^{m} \mu_{\mathcal{G}}(\mathcal{R}_i) \\
&= p.\sum_{i=1}^{m} \mu_{\mathcal{G}}(\mathcal{R}_i).\sum_{j=1}^{\infty} \mu_{\mathcal{G}}(\emptyset, V, \iota)^{j-1} \\
&= p.\sum_{i=1}^{m} \mu_{\mathcal{G}}(\mathcal{R}_i).\frac{1}{1-\mu_{\mathcal{G}}(\emptyset, V, \iota)} \text{ by standard results on geometric series} \\
&= p.(1-\mu_{\mathcal{G}}(\emptyset, V, \iota)).\frac{1}{1-\mu_{\mathcal{G}}(\emptyset, V, \iota)} \\
&= p
\end{aligned}$$

Hence, for every $A \in \mathit{Adv}_{\mathcal{M}_{\mathcal{G}}}$, $\mathit{Prob}^{A}_{s_0}(\{\omega \in \mathit{Path}^{A}_{s_0} | \omega \models \psi\}) \bowtie p$ and therefore, $\mathcal{M}_{\mathcal{G}} \models \phi$.

Let ϕ be an unindexed QLTL$_{\backslash\mathcal{X}}$ formula with propositions in $\bigcap_{\mathcal{G}\in\Gamma} \bigcap_{\mathcal{R}\in \mathit{support}(\mu_{\mathcal{G}})} AP'_{\mathcal{R}(\mathcal{G})}$ and let $\mathcal{G} \in \Gamma$. If $\mathcal{G} \in \Phi$ then, by the statement of the theorem, $\mathcal{M}_{\mathcal{G}} \models \phi$. Otherwise, $\mathcal{G} \in \Gamma \setminus \Phi$. Since ϕ is not indexed by any process index, and ϕ is defined over $\bigcap_{\mathcal{R}\in \mathit{support}(\mu_{\mathcal{G}})} AP'_{\mathcal{R}(\mathcal{G})}$, by the above, $\mathcal{M}_{\mathcal{G}} \models \phi$ if $\mathcal{M}_{\mathcal{R}(\mathcal{G})} \models \phi$ for every complete reduction, $\mathcal{R} \in \mathit{support}(\mu_{\mathcal{G}})$). For each complete reduction, $\mathcal{R} \in \mathit{support}(\mu_{\mathcal{G}})$), either $\mathcal{R}(\mathcal{G})$ is in Φ or it can be reduced further. Continuing in this way, we will reach a graph in Φ (since Γ is reducible to Φ with probability 1 under $\mathcal{Q}_{\mathcal{G}}$). Finally, by the statement of the theorem, each of the models associated with the graphs in Φ satisfy ϕ and it follows that $\mathcal{M}_{\mathcal{G}} \models \phi$. □

6.2 Case Study: The Itai Rodeh Leader Election Protocol

6.2.1 Introduction

To illustrate our technique, we consider the PMCP for a particular randomised distributed protocol designed to elect a leader from a set of processes arranged in a ring, which we will demonstrate later is probabilistically degenerative. More specifically, we analyse the asynchronous version of the Itai Rodeh leader election protocol (IRP) [35]. This protocol has already been analysed, using PRISM, for fixed size systems [1]. In addition, some formal analysis has been carried out for the Itai Rodeh leader election protocol, by hand. However, we are not aware of any work to date that attempts to tackle the parameterised model checking problem for these protocols using a similar approach to ours. The protocol is described in full in [35], we give an overview below.

We first note that it has been shown for an anonymous ring of processes (i.e. where the process id's are unknown) that are identical up to renaming, there is no deterministic algorithm that can elect a unique leader [3]. However, if we loosen the condition 'a unique leader must be chosen' to 'a unique leader must be chosen with probability 1' then it is possible to employ a probabilistic algorithm to break the symmetry of the processes and allow a unique leader to be elected (assuming we know the size of the ring). This is the basis of the IRP.

The algorithm proceeds in two phases. Initially all processes are deemed *active*. In the first phase a process selects 0 or 1, each with probability a half. Having chosen its preference, a process can then send its selection to the next process in the ring (the ring being unidirectional). When a process receives its neighbours choice, if it is 1 and its own choice was 0 then the process will become *passive* and will then only pass on messages that it receives (although incrementing any counter it receives, see below). For any other combination of choices the process remains active and moves to phase two.

In the second phase, any active process transmits a counter, initialised to 0, that is passed around the ring. Each time this counter passes through a passive process it is incremented by one. Therefore, if an active process receives a counter with value $N-1$ (where N is the size of the ring) it must be the only remaining active process and so can declare itself leader. On the other hand, if an active process receives a counter with value less than $N-1$ it cannot be unique and so will proceed to phase 1 and select its preference again. This continues until a leader is chosen.

Note that in practice, although we describe the protocol as proceeding in phases, in reality, because the system is asynchronous, the processes do not execute phases together, so for example one process may be in phase one while another has completed phase two. The sequence of actions, however, for a single process is as described above.

6.2.2 Parameterised Models of the IRP

The IRP has been modelled and verified (for rings of a fixed size) using the PRISM probabilistic model checker [1,30]. We have modified these PRISM definitions to model the IRP using asynchronous communication with neighbouring processes transmitting along channels of length N, where N is the size of the ring. As an example, a PRISM definition for a ring of size three is given in Appendix F.

We let Γ denote the set of rings (of size greater than one), with the vertices of each graph $\mathcal{G}$ in Γ labelled by the values $0, 1 \ldots, |\mathcal{G}|-1$. Also, we let $\Phi = \{\mathcal{G}(2)\}$ where $\mathcal{G}(2)$ is the ring of size two. We have written a script program that can generate PRISM specifications of the IRP for every member of Γ, based on PRISM specifications that we defined for rings up to size 5. We denote the set of specifications that can be generated by this script by $\mathcal{S}(\Gamma) = \{\mathcal{S}(\mathcal{G})|\mathcal{G} \in \Gamma\}$ and the family of models over $\mathcal{S}(\Gamma)$ by $\mathbf{M}_\Gamma = \{\mathcal{M}_\mathcal{G}|\mathcal{G} \in \Gamma\}$.

For $\mathcal{G} = (E, V, I) \in \Gamma$, the set of variables associated with $\mathcal{S}_\mathcal{G}$ is the set $X_\mathcal{G} = \cup_{i=0}^{N-1} X^i_\mathcal{G} \cup C_\mathcal{G} \cup G_\mathcal{G}$ where $X^i_\mathcal{G}$ denotes the set of local variables of process i, $G_\mathcal{G}$ the set of global variables and $C_\mathcal{G}$ the set of channel variables with, for $0 \leq i \leq N-1$,

$$
\begin{aligned}
&X^i_\mathcal{G} = \{state_i, sent_i, receive_i, p_i, c_i\},\\
&C_\mathcal{G} = \{ch_{i,j}|(i,j) \in E\}\\
&G_\mathcal{G} = \{leader\}
\end{aligned}
$$

The variable domains are given by, for $0 \leq i \leq N-1$, $ch_{i,j} \in C_\mathcal{G}$,

$$
\begin{aligned}
&D(state_i) = \{active, passive, leader, finish\},\\
&D(sent_i) = \{0, 1, 2\},\\
&D(receive_i) = \{0, 1, 2\},\\
&D(c_i) = \{0, 1, \ldots, N-1\},\\
&D(p_i) = \{0, 1, 2\},\\
&D(leader) = \{\perp, 0, 1, \ldots, N-1\},\\
&D(ch_{i,j}) = \{empty, 0, 1, \ldots, N-1\}.
\end{aligned}
$$

We let $Act^\mathcal{G}$ denote the set of actions associated with $\mathcal{M}_\mathcal{G}$.

For communication graph, $\mathcal{G}$, a ring of size N the corresponding PRISM definition $\mathcal{S}(\mathcal{G})$ describes modules for the set of processes in the ring and for the set of channels. We

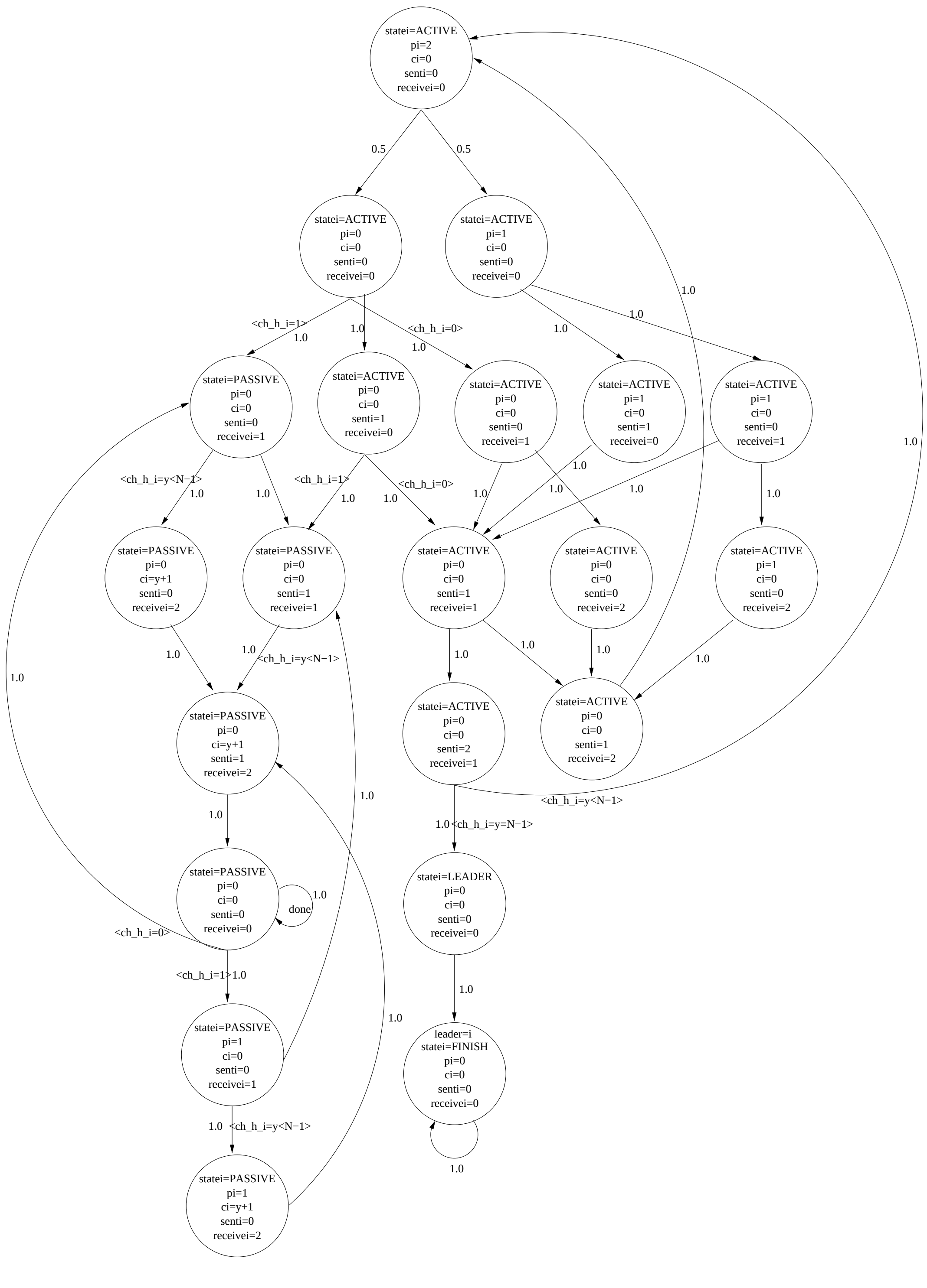

statei=ACTIVE pi=2 ci=0 senti=0 receivei=0
0.5
0.5
statei=ACTIVE pi=0 ci=0 senti=0 receivei=0
statei=ACTIVE pi=1 ci=0 senti=0 receivei=0
<ch_h_i=1>
<ch_h_i=0>
<ch_h_i=y<N−1>
<ch_h_i=y=N−1>
statei=PASSIVE pi=0 ci=0 senti=0 receivei=1
statei=ACTIVE pi=0 ci=0 senti=1 receivei=0
statei=ACTIVE pi=0 ci=0 senti=0 receivei=1
statei=ACTIVE pi=1 ci=0 senti=1 receivei=0
statei=ACTIVE pi=1 ci=0 senti=0 receivei=1
statei=PASSIVE pi=0 ci=y+1 senti=0 receivei=2
statei=PASSIVE pi=0 ci=0 senti=1 receivei=1
statei=ACTIVE pi=0 ci=0 senti=1 receivei=1
statei=ACTIVE pi=0 ci=0 senti=0 receivei=2
statei=ACTIVE pi=1 ci=0 senti=0 receivei=2
statei=PASSIVE pi=0 ci=y+1 senti=1 receivei=2
statei=ACTIVE pi=0 ci=0 senti=2 receivei=1
statei=ACTIVE pi=0 ci=0 senti=1 receivei=2
statei=PASSIVE pi=0 ci=0 senti=0 receivei=0
done
statei=LEADER pi=0 ci=0 senti=0 receivei=0
statei=PASSIVE pi=1 ci=0 senti=0 receivei=1
leader=i statei=FINISH pi=0 ci=0 senti=0 receivei=0
statei=PASSIVE pi=1 ci=y+1 senti=0 receivei=2
1.0

Figure 6.1: State graph for a single process, $process_i$, in a model, $\mathcal{M}_{\mathcal{G}}$, of the Itai Rodeh protocol

consider the module specification for $process_i$ $(0 \leq i \leq |\mathcal{G}| - 1$. The variable $state_i$ represents the state of process i. The variable $sent_i$ stores a value from $\{0, 1, 2\}$ where 0 indicates that no messages have been sent, 1 means that process i has sent its preference and 2 that process i has also sent its counter. Similarly $receive_i$ has domain $\{0, 1, 2\}$ and if $receive_i = 0$ then process i has received no messages, if $receive_i = 1$ then process i has received its neighbours preference and if $receive_i = 2$ then process i has also received its neighbours counter. The variable c_i stores the value of any counter that process i receives. When process i chooses its preference this is stored in p_i which has value 2 before the process has chosen its preference or preference value 0 or 1 once chosen. Also, if $state_i = passive$, p_i also stores the value of any preference it receives from its neighbour. The modules for all other processes have the same local variables as $process_i$ but are renamed with the process id.

The unique global variable, $leader$, has an indexed domain and is set to the value of a process index if a process is elected leader. We assume that $process_h$ is the left neighbour of $process_i$ and $process_j$ is the right neighbour $(h \neq i,\ i \neq j,\ h \neq j,\ 0 \leq j, h \leq |\mathcal{G} - 1|)$.

The module for the channel from process i to process j (channel i, j) has local variables $ch^1_{i,j}, ch^2_{i,j}, \ldots, ch^N_{i,j}$, representing each of the positions in a channel of length N. Each local variable may store the value of a process's preference or a process's counter and therefore has a value in the range $empty, 0, 1, \ldots, N-1$. Note that $empty$ is a default value used to initialise the variables and to represent the fact that nothing is stored in that position in the channel. The channel is FIFO and so whenever an item is sent on the channel it is stored in the first free position. If the channel is full, i.e. if $ch^1_{i,j}$ is not equal to $empty$, then the send cannot occur. Similarly, whenever process j receives a message from channel i, j, this will be read from $ch^1_{i,j}$ and all other messages will be 'shifted' one place up the channel i.e. the message in $ch^2_{i,j}$ will be moved into $ch^1_{i,j}$, the message in $ch^3_{i,j}$ will be moved into $ch^2_{i,j}$ etc. If the channel is empty i.e. $ch^1_{i,j} = empty$ then process j cannot perform a read. For convenience, we use $ch_{i,j}$ to denote channel i, j and let $ch_{i,j}[n] = ch^n_{i,j}$. The variables for all other channels are the same as for channel i, j up to renaming of the channel index.

Initially, $process_i$ will have $state_i = active$, $sent_i = 0$, $receive_i = 0$, $c_i = 0$, $p_i = 0$ and similarly for all the other process modules. Also, initially $ch_{h,i} = ch_{i,j} = [empty, \ldots, empty]$ (and similarly for all channel modules). From this state, the only executable statement for $process_i$ sets sets $p_i = 0$ or $p_i = 1$ with equal probability.

Having chosen a preference, $process_i$ can then either receive its neighbours preference if this is available from its in channel or it can send its own preference on its out channel. If $process_i$ sends its preference then $ch_{i,j}[1]$ is set to the value of p_i and $sent_i$ is set to 1. From this state it is only possible for $process_i$ to read its neighbours preference. Whenever $process_i$ reads its neighbours preference from its in channel then it sets $receive_i = 1$ and if $p_i = 0$ and $ch_{h,i}[1] = 1$ then $state_1 = passive$, otherwise $state_i = active$. First we assume that $state_i = active$. If $sent_i = 0$ then $process_i$ can send its preference, setting $sent_i = 1$ and sending p_i on its out channel. If $sent_i = 1$ and $receive_i = 1$ then $process_i$ can send its counter (which will have value 0), setting $sent_i = 2$. If $sent_i = 1$ and $receive_i = 2$ then $process_i$ must have received a counter from another active process. Hence, once it has sent its counter $process_i$ must choose another preference and so we set $sent_i = 0$, $receive_i = 0$, $c_i = 0$ and $p_i = 0$ and can then proceed as described above.

When $state_i = active$, if $receive_i = 1$ then $process_i$ can receive its neighbours counter. In this case, if $sent_i < 2$, i.e. $process_i$ has not yet sent its counter then this must be the counter of another process and $receive_i$ is set to 2. If, on the other hand, $sent_i = 2$ and $process_i$ receives a counter with value less than $N - 1$ then there must be other active processes present in the ring and it is necessary to choose again. Thus, $state_i = active$, $sent_i = 0$, $receive_i = 0$, $c_i = 0$, $p_i = 0$ and the protocol proceeds as described above.

If, however, $process_i$ receives a counter value of $N-1$ then $process_i$ must be the only active process in the ring and so the local variables are set so that $state_i = leader$, $sent_i = 0$, $receive_i = 0$, $c_i = 0$ and $p_i = 0$. From this state, the assignments are made such that $leader = i$ and $state_i = finish$ and the protocol terminates.

Assume that $p_i = 0$ and $process_i$ has received its neighbours choice which is equal to 1 so that $state_i = passive$. In this case, $process_i$ can then receive its neighbours counter if this is available. Assuming that value C is received then c_i is set to $C + 1$ and $receive_i$ is set to 2.

If $state_i = passive$ and $sent_i = 0$ then $process_i$ can send its preference, setting $sent_i = 1$. Having sent its preference, if $process_i$ has received its neighbours counter i.e. $receive_i = 2$ then $process_i$ can pass on the counter, setting $sent_i = 0$, $receive_i = 0$, $c_i = 0$, $p_i = 0$. From this state, $process_i$ can only receive its neighbours preference, in which case it sets $receive_i = 1$. The process will continue to behave as described above until a process is elected leader. Note that the behaviour of any of the process or channel modules will be identical to that described above.

Figure 6.1 gives the local state graph associated with $process_i$. Note that only the variables of $process_i$ are included in the labelling of the states and we omit action labels (and associated 'dashed edges') for clarity of presentation. We use the notation $< g >$ to indicate a guard g that must be true in order for a transition to be taken. It is assumed, however, that guards referring to the state of the channels (a send can only take place if a channel is not full, a read can only take place if a channel is not empty) are implicit.

Figure 6.2 also gives an example execution for the PRISM specification for a ring of size three. In the figure, we abbreviate the values *active*, *passive*, *leader* and $finish$ by A, P, L and F respectively. The value of the channel variables are given in the square brackets ([]). We do not use the default value of *empty* to indicate an empty position but instead only show values that are sent on the channel. The first value between the square brackets is the first to be read from the channel.

6.2.3 Model checking the IRP using PRISM

The table in Figure 6.2.3 gives the results of building and then verifying (with respect to Property 1, below) a model in PRISM from our definition for rings of size $N = 2, 3, 4, 5$. It displays the time taken to build the model, the number of states of the MDP and the number of nodes in the MTBDD. The last column also gives the time taken to verify the property defined below (where *leader* represents the global boolean variable of the PRISM specification that is initially equal to zero and is set to the value of the process index when that process is elected leader). Note that *Property 1* is an unindexed $\text{QLTL}_{\backslash\mathcal{X}}$ property.

Property 1 A leader is elected with probability 1.

$$\mathcal{P}_{\geq 1}[\Diamond\neg(leader = \bot)]$$

6.2.4 Parameterised model checking of the IRP

Having verified the Itai Rodeh leader election protocol for fixed sizes of ring using the PRISM model checking tool, we now tackle the problem of proving properties for an arbi-

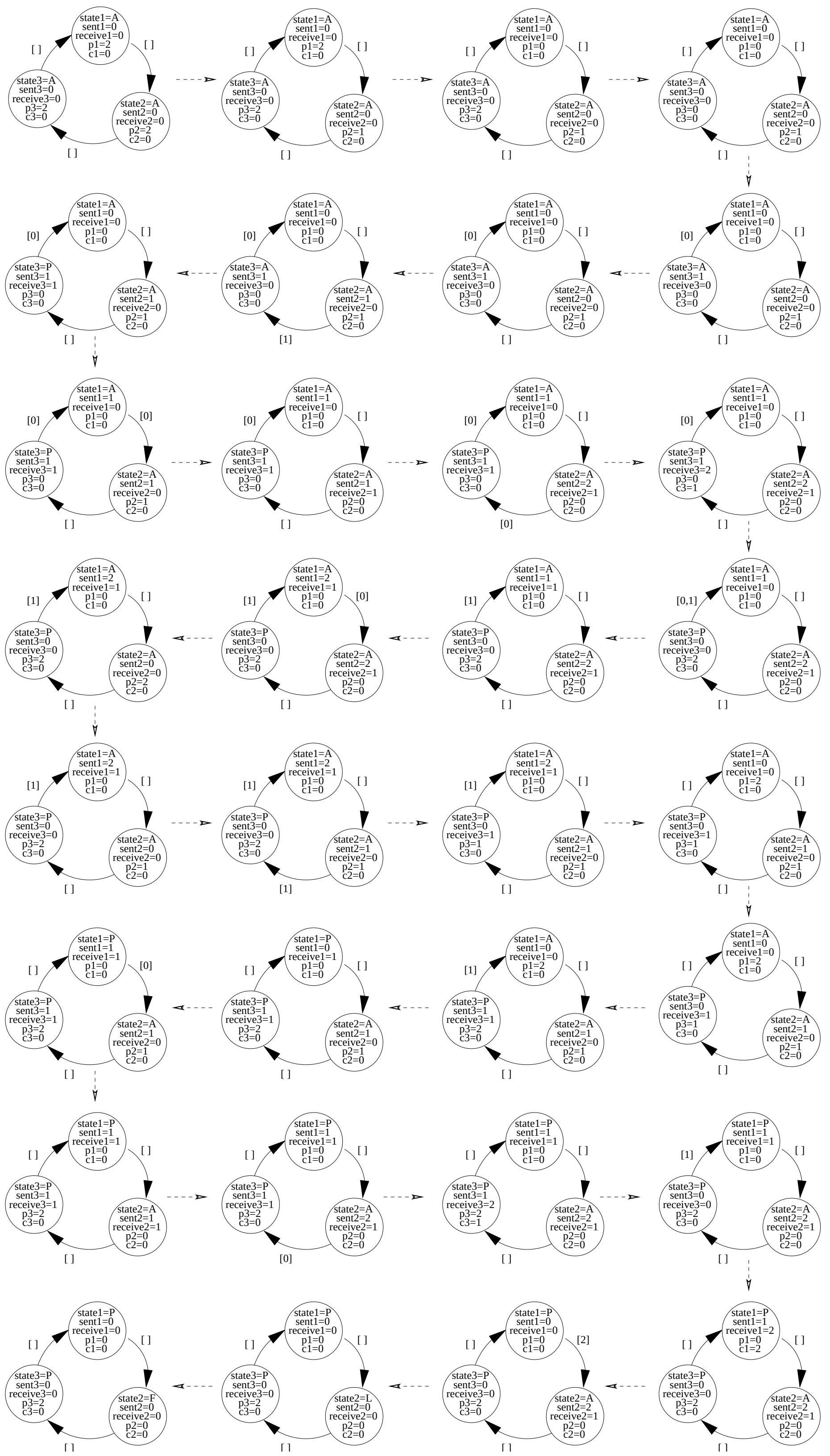


Figure 6.2: Example execution of the PRISM specification of the Itai Rodeh protocol for a ring of size three

Table 6.1: Model sizes and build and verification times

N	Time(s)	States	Nodes	Property 1
2	0.077	91	1882	0.052
3	0.405	994	10874	0.609
4	1.956	12177	50297	5.248
5	11.370	150507	181253	37.510

trary size of ring. In order to do so, we can exploit the proof technique for *probabilistically degenerative* systems described above.

We aim to show that the Itai Rodeh protocol is probabilistically degenerative. Intuitively, for a ring of some size, once a set of processes become passive then the protocol behaves like a smaller ring of processes. However, it cannot be described as deterministically degenerative as for the Firewire protocol since a passive process is chosen probabilistically. Note also that a passive process can still pass on messages and increment the value of any counter it receives (the maximum value of which is dependent on the size of the ring). Nonetheless, it is still possible to employ an inductive proof in the manner described above if we restrict the set of propositions that we consider. Specifically, for the models $\mathcal{M}_{\mathcal{G}} \in \mathbf{M}_{\Gamma}$ of the IRP, for a ring of size greater than or equal to two, we intend to prove the following.

Theorem 6.2.1. *Let ϕ be Property 1, then for any $\mathcal{G} \in \Gamma$, $\mathcal{M}_{\mathcal{G}}$ satisfies ϕ.*

Using the methodology described in Section 6.1.2 above, we prove properties of our *base* family of models ($\{\mathcal{M}_{\mathcal{G}(2)}\}$) by model checking and then for $N > 2$, establish a relationship between $\mathcal{M}_{\mathcal{G}}$ and $\mathcal{M}_{\mathcal{G}'}$ for each reduced graph $\mathcal{G}'$ of $\mathcal{G}$ under a particular type of incomplete cyclic reduction.

Lemma 6.2.2. *For all $\mathcal{G} \in \Phi$, $\mathcal{M}_{\mathcal{G}} \models \phi$, where ϕ is Property 1.*

Proof. By model checking (see entry for $N = 2$ in Table 6.2.3). □

In a similar manner as for the TIP, we show that $\mathbf{M}_{\Gamma}$ is (probabilistically) degenerative with base Φ by considering each of the conditions given in Definition 6.1.6 in turn, having defined an appropriate set of reductions and corresponding set of variables for each model in $\mathbf{M}_\Gamma$.

Join Reductions

We define a family of distributions $\mathcal{Q}_\Gamma$ over sets of reductions on graphs in Γ.

Definition 6.2.3. *Given $\mathcal{G} = (E, V, I) \in \Gamma$ and $0 \leq i \leq N-1$, let disjoint$^i(\mathcal{G})$ denote the family of sets of i vertices of $\mathcal{G}$ such that no two vertices in a set in disjoint$^i(\mathcal{G})$ has an edge between them. Furthermore, let join$^i(\mathcal{G})$ be the family of sets of vertices of $\mathcal{G}$ such that, for each $\{k_1, k_2, \ldots, k_i\} \in$ disjoint$^i(\mathcal{G})$, $V \setminus \{k_1, k_2, \ldots, k_i\} \in$ join$^i(\mathcal{G})$.*

Note that for $\mathcal{G} = (E, V, I) \in \Gamma$, *disjoint*$^0(\mathcal{G}) = \{\emptyset\}$ and therefore *join*$^0(\mathcal{G}) = \{V\}$.

Lemma 6.2.4. *If $\mathcal{G}$ is a ring, then for $\lfloor N/2 \rfloor < i \leq N$, disjoint$^i(\mathcal{G}) = \emptyset$ and for $1 \leq i \leq \lfloor N/2 \rfloor$, $|$disjoint$^i(\mathcal{G})| \geq 1$.*

Proof. Let $\mathcal{G} \in \Gamma$ for $|\mathcal{G}| = N$. Define K as the set of all vertices of $\mathcal{G}$ that are labelled by an odd value. It should be apparent that there are no bigger sets of disjoint vertices of $\mathcal{G}$: adding any vertex to K would result in a sequence of vertices that are not disjoint. For N even, $|K| = N/2$ and for N odd, $|K| = (N-1)/2$. For $1 \leq i \leq |K|$, a set of disjoint vertices of size i can be derived by removal of vertices from K. $\square$

In the sequel, we let $\sigma^{\mathcal{G}}_{J^i}$ be a permutation (for $\lfloor N/2 \rfloor \geq i \geq 0$, $J^i \in$ *joint*$^i(\mathcal{G})$) on the vertex labels of $\mathcal{G} = (E, V, I) \in \Gamma$, which permutes the indices such that the vertices of $\mathcal{G}$ in $V \setminus J^i$ have the largest indices and the order of the indices of the vertices in J^i is preserved.

Definition 6.2.5. *Let $\mathcal{G} = (E, V, I) \in \Gamma$ with $|\mathcal{G}| = N$ and for $\lfloor N/2 \rfloor \geq i \geq 0$ let $J^i = \{j_0, j_1, \ldots, j_{N-1-i}\} \in$ joint$^i(\mathcal{G})$ with $j_t < j_{t+1}$ for $0 \leq t < N-1-i$. Let $\bar{J}^i = V \setminus J^i = \{k_0, \ldots, k_{i-1}\}$ where $k_u < k_{u+1}$ for $0 \leq u < i-1$. Define the bijective map $\sigma^{\mathcal{G}}_{J^i}$ on the set $\{\perp, 0, 1, 2, \ldots, N-1\}$ such that $\sigma^{\mathcal{G}}_{J^i}(\perp) = \perp$, $\sigma^{\mathcal{G}}_{J^i}(j_t) = t$ for all $0 \leq t \leq N-1-i$ and $\sigma^{\mathcal{G}}_{J^i}(k_u) = u + N - i$ for all $0 \leq u \leq i-1$.*

Furthermore, because we want to reduce to a new ring, we must identify a set of edges that will close the subgraph induced by any subset of the vertices of a ring in *join*$^i(\mathcal{G})$.

Definition 6.2.6. *Let V_{J^i} be the subset of the set of vertices in a set J^i in $join^i(\mathcal{G})$ (for $\mathcal{G} \in \Gamma$, $\lfloor N/2 \rfloor \geq i \geq 0$) such that for each $v \in V_{J^i}$ there is no outgoing edge from v in the graph $\mathcal{G}[J]$. Similarly, let U_{J^i} be the set of vertices such that for each $u \in U_{J^i}$ there is no incoming edge to u in the graph $\mathcal{G}[J^i]$. For $v_0 \in V_{J^i}$, $u \in U_{J^i}$, let $(v_0, u) \in F^{\mathcal{G}}_{J^i}$ if and only if there exists edges $(v_0, v_1), (v_1, v_2), \dots, (v_{n-1}, v_n), (v_n, u)$ for some $N \geq n \geq 1$ such that for each $0 \leq i \leq n-1$, (v_i, v_{i+1}) is in the edge set of $\mathcal{G}$ but not in the edge set of $\mathcal{G}[J^i]$, and similarly for (v_n, u).*

Definition 6.2.7. *For $\mathcal{G} \in \Gamma \setminus \Phi$, define a set of* join *reductions on $\mathcal{G}$ as*

$$Join^{\mathcal{G}} = \{(F^{\mathcal{G}}_{J^i}, J^i, \sigma^{\mathcal{G}}_{J^i}) | \text{ for all } J^i \in join^i(\mathcal{G}) \text{ for every } \lfloor N/2 \rfloor \geq i \geq 0\}.$$

Note that for $i = 0$ in Definition 6.2.7, the unique join reduction $(F^{\mathcal{G}}_{J^0}, J^0, \sigma^{\mathcal{G}}_{J^0})$ is equivalent to the identity reduction. An example of a graph obtained under a join reduction for $N = 5$ and $i = 2$ is shown in Figure 6.3.

Let $\mathcal{G} \in \Gamma$ (with $|\mathcal{G}| = N$ and let $D^i = \{j_0, j_1, \dots, j_{i-1}\} \in disjoint^i(\mathcal{G})$ with $j_{t-1} < j_t$ for $1 \leq t \leq i-1$. We want the probability that in the first round of the Itai-Rodeh protocol, every process in D^i becomes passive and all other processes remain active. In order for this to happen we must have a sequence of preference choices (assuming that j_0's preference is the first in the sequence) of the form

$$(0^{z_0}1^{y_0})(0^{z_1}1^{y_1})\dots(0^{z_{i-1}}1^{y_{i-1}}) \text{ for } z_r, y_r \geq 1, z_r + y_r = d_r \tag{6.2}$$

where d_r is the number of processes between each pair of processes j_r, $j_{r\oplus_i 1}$ in D^i. The number of sequences of the form (0^z1^y), $z, y \geq 1$ is given by $z + y - 1$. Therefore, the total number of sequences of the form of 6.2 is given by

$$\prod_{r=0}^{i-1} d_r - 1$$

and hence the probability of the processes in D^i becoming passive while the remaining processes remain active is given by

$$\frac{\prod_{r=0}^{i-1} d_r - 1}{2^N}.$$

Definition 6.2.8. *For $\mathcal{G} \in \Gamma$ let $\mu_{\mathcal{G}} : Join^{\mathcal{G}} \rightarrow [0, 1]$ be a distribution over the set of join reductions of $\mathcal{G}$ such that, for the ring of size 2, $\mathcal{G}(2)$,*

$$\mu_{\mathcal{G}(2)}((\emptyset, V, \iota)) = 1$$

and for $\mathcal{G} \in \Gamma \setminus \Phi$,

$$\mu_{\mathcal{G}}((\emptyset, V, \iota)) = \frac{2}{2^N},$$

and for every $J^i \in disjoint^i(\mathcal{G})$, for all $\lfloor N/2 \rfloor \geq i \geq 0$,

$$\mu_{\mathcal{G}}((F_{\mathcal{G}}^{J^i}, J^i, \sigma_{J^i}^{\mathcal{G}})) = \frac{\prod_{r=0}^{i-1} d_r - 1}{2^N}.$$

We can now define a set of reductions on a graph $\mathcal{G} \in \Gamma$.

Definition 6.2.9. *Define the set of* join reductions *over Γ as*

$$\mathcal{Q}_\Gamma = \{\mu_{\mathcal{G}} | \mathcal{G} \in \Gamma\}$$

Essentially, a join reduction is a reduction in which a covering set of vertices is removed, the removing vertices are relabelled to have the smallest index values and edges are added to preserve the ring. We can show that the set of join reductions is reducible to the singleton set containing the ring of size two, with probability 1.

Lemma 6.2.10. *The set $\mathcal{Q}_\Gamma$ is reducible to Φ with probability 1.*

Proof. For all $\mathcal{G} \in \Gamma$, construct an infinite state DTMC $\mathcal{G} = (\Gamma, \Gamma, \mathbf{P}, L)$ over $G = \{g\}$ where, for $\Delta, \Delta' \in \Gamma$,

$$\mathbf{P}(\Delta, \Delta') = \sum_{\mathcal{R} \in support(\mu_\Delta), \Delta' = \mathcal{R}(\Delta)} \mu_\Delta(\mathcal{R}),$$

and $D(g) = \{0, 1\}$ with, for all $\mathcal{G} \in \Gamma$, $L(\mathcal{G}) = \{g = 1\}$ if $\mathcal{G} \in \Phi$, $L(\mathcal{G}) = \{g = 0\}$, otherwise.

We prove, by induction on the size of a graph that for all $\mathcal{G}$, $\mathcal{G} \models \mathcal{P}_{\geq 1}[\Diamond g = 1]$.

Base Case: Let $\mathcal{G} \in \Gamma$ be the ring of size two. Then by definition of the labelling function of G, $g = 1 \in L(\mathcal{G})$ and thus $\mathcal{G}_{\mathcal{G}} \models \mathcal{P}_{\geq 1}[\Diamond g = 1]$.

Induction Step: Suppose that, for all $\mathcal{F} \in \Gamma$ with $|\mathcal{F}| = N$ for $N \geq 2$, $\mathcal{F} \models \mathcal{P}_{\geq 1}[\Diamond g = 1]$. Let $\mathcal{G} \in \Gamma$ with $|\mathcal{G}| = N + 1$. Let $\mu_{\mathcal{G}} \in \mathcal{Q}_\Gamma$ be the distribution over the join reductions on $\mathcal{G}$. Since $|\mathcal{G}| \geq 3$, there must be at least one complete join reduction in *support*$(\mu_{\mathcal{G}})$. Note that for any complete join reduction of $\mathcal{G}$, the reduced graph $\mathcal{F}$ will be such that $2 \leq |\mathcal{F}| \leq N$. Let Δ be the set of reduced graphs of $\mathcal{G}$ under some *complete* join reduction in *support*$(\mu_{\mathcal{G}})$ then clearly the probability of reaching a state in Δ from $\mathcal{G}$ is 1. By the

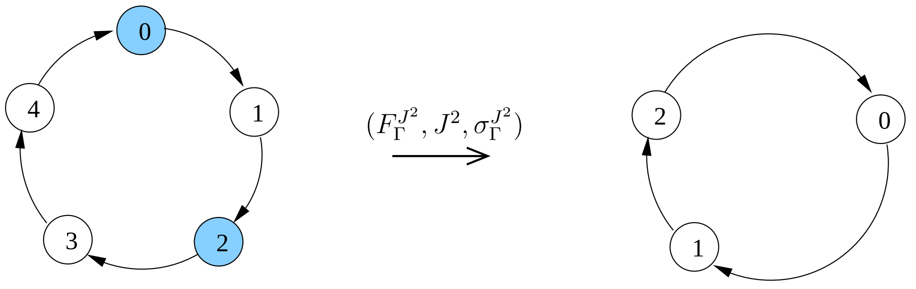


Figure 6.3: An example of graph $\mathcal{G}$ (left) and the graph $\sigma_{\mathcal{G}}^{J^2}(\mathcal{G})[J^2] \oplus F_{\mathcal{G}}^{J^i}$ (right) obtained under a join reduction where $J^2 = \{1, 3, 4\}$.

induction hypothesis, the probability of reaching a state with $g = 1$ from a state in Δ is equal to 1 and therefore, $\mathcal{G} \models \mathcal{P}_{\geq 1}[\Diamond g = 1]$.

By induction, Γ is therefore reducible to Φ with probability 1 under $\mathcal{Q}_\Gamma$.

□

We now define a subset of the variable set of a model of a join reduced graph.

Definition 6.2.11. *Given $\mathcal{M}_{\mathcal{G}}$ over $X_{\mathcal{G}}$ and a join reduction, $\mathcal{R} = (F_i^{\mathcal{G}}, J^i, \sigma_i^{\mathcal{G}})$, define,*

$$\mathbf{X}'_{\mathcal{R}(\mathcal{G})} = \{leader\} \cup \cup_{j \in J^i}\{state_j\}.$$

Define $AP'_{\mathcal{R}(\mathcal{G})}$ to be the set of atomic propositions over $\mathbf{X}'_{\mathcal{R}(\mathcal{G})}$.

Now we show how $\mathcal{M}_\Gamma$, with join reductions, is probabilistic degenerative, by demonstrating that each of the conditions of Definition 6.1.6 is fulfilled.

Condition 1 (Reduced Variables and Actions)

Lemma 6.2.12. *Given a graph $\mathcal{G} \in \Gamma$ that is not the ring of size two and a complete join reduction, $(F_{\mathcal{G}}^{J^i}, J^i, \sigma_{\mathcal{G}}^{J^i})$ then,*

$$X_{\mathcal{G}} \setminus C_{\mathcal{G}} = X_{\sigma_{\mathcal{G}}^{J^i}(\mathcal{G})} \setminus C_{\mathcal{G}}, D(X_{\mathcal{G}}) = D(X_{\sigma_{\mathcal{G}}^{J^i}(\mathcal{G})}), Act_{\mathcal{G}} = Act_{\sigma_{\mathcal{G}}^{J^i}(\mathcal{G})}.$$

Lemma 6.2.13. *Given a graph $\mathcal{G} \in \Gamma$ that is not the ring of size two and a complete join reduction, $Join_{\mathcal{G}}^{J^i} = (F_{\mathcal{G}}^{J^i}, J^i, \sigma_{\mathcal{G}}^{J^i})$ then,*

$$X_{Join_{\mathcal{G}}^{J^i}(\mathcal{G})} \subseteq X_{\sigma_{\mathcal{G}}^{J^i}(\mathcal{G})}, D(X_{Join_{\mathcal{G}}^{J^i}(\mathcal{G})}) \subseteq D(X_{\sigma_{\mathcal{G}}^{J^i}(\mathcal{G})}), Act_{Join_{\mathcal{G}}^{J^i}(\mathcal{G})} \subseteq Act_{\sigma_{\mathcal{G}}^{J^i}(\mathcal{G})}.$$

The proofs of Lemmas 6.2.12 and 6.2.13 follow from the definition of the variable sets, variable domains and action sets over $\mathcal{S}(\Gamma)$ and from the definition of the join reductions given previously.

Condition 2 (Matching Adversaries)

We partition the paths of the adversaries of an IRP model according to their behaviour with respect to the processes that become passive. Specifically, we classify them according to the processes that choose zero and their left neighbour chooses 1 in the first round. If j is such a vertex then the value of variable $state_j$ will change from *active* to *passive* when it receives its neighbours choice. Process j is then guaranteed to remain passive and its effect under the adversary can be ignored.

Definition 6.2.14. *Let $\mathcal{G} \in \Gamma \setminus \Phi$ and let $J^i \in join^i(\mathcal{G})$ for some $1 \leq i \leq \lfloor N/2 \rfloor$. For an adversary $A \in Adv_{\mathcal{M}_\mathcal{G}}$, a path $\omega \in Path^A_{s_0}$ is said to be* first-full *with respect to J^i iff,*

$$\begin{aligned} \omega \models \quad & \wedge_{j \in J^i} \quad receive_j = 0\, \mathcal{U}\, (receive_j = 1 \wedge state_i = PASSIVE) \\ & \wedge_{k \notin J^i} \quad receive_k = 0\, \mathcal{U}\, (receive_k = 1 \wedge state_k = ACTIVE). \end{aligned}$$

Define $P^A_{s_0}(J^i)$, the set of paths under A that are first-full w.r.t. J^i.

Lemma 6.2.15. *Let $\mathcal{G} \in \Gamma$ and let $A \in Adv_{\mathcal{M}_\mathcal{G}}$. The set*

$$\{P^A_{s_0}(J^i) | J^i \in disjoint^i(\mathcal{G}), 0 \leq i \leq \lfloor N/2 \rfloor\}$$

is a measurable partition of $Path^A_{s_0}$.

Proof. For each $0 \leq i \leq \lfloor N/2 \rfloor$, for $J^i \in disjoint^i(\mathcal{G})$, the set of paths $P^A_{s_0}(J^i)$ is described by a LTL property and is therefore measurable.

The set of paths $P^A_{s_0}(J^i)$ represents the paths under which exactly the J^i processes become passive in the first round. For any K^j, $0 \leq j \leq \lfloor N/2 \rfloor$, $K^j \neq J^i$, it is not possible to choose the K^j processes as passive in the first round as well as the J^i processes. Therefore $P^A_{s_0}(J^i) \cap P^A_{s_0}(K^j) = \emptyset$.

In the first round, either the J^i processes are chosen to become passive for some $J^i \in disjoint^i(\mathcal{G})$, $1 \leq i \leq \lfloor N/2 \rfloor$ or no processes are selected to become passive. Therefore, $Prob^A_{s_0}(\bigcup_{0 \leq i \leq \lfloor N/2 \rfloor} P^A_{s_0}(J^i)) = 1.$ □

Table 6.2: Result of applying $\Sigma^{\mathcal{G}}_{J^i}$ to an atomic proposition, a, for $0 \leq h, i \leq N$ (where σ is an abbreviation of $\sigma^{\mathcal{G}}_{J^i}$)

a	$\Sigma^{\mathcal{G}}_{J^i}(a)$
$state_h = x$	$state_{\sigma(h)} = x$
$sent_h = x$	$sent_{\sigma(h)} = x$
$receive_h = x$	$receive_{\sigma(h)} = x$
$c_h = x$	$c_{\sigma(h)} = x$
$p_h = x$	$p_{\sigma(h)} = x$
$leader_h = x$	$leader_{\sigma(h)} = \sigma(x)$
$ch_{h,i} = x$	$ch_{\sigma(h),\sigma(i)} = x$

.

Let $\mathcal{G} \in \Gamma \setminus \Phi$ and for some $J^i \in join^i(\mathcal{G})$, let $Join^{J^i}_{\mathcal{G}} = (F^{J^i}_{\mathcal{G}}, J^i, \sigma^{\mathcal{G}}_{J^i})$ be a join reduction of $\mathcal{G}$. Let $AP_{\mathcal{G}}$ be the set of atomic propositions over $X_{\mathcal{G}}$ and let $AP_{\sigma^{\mathcal{G}}_{J^i}(\mathcal{G})}$ be the set of atomic propositions over $X_{\sigma^{\mathcal{G}}_{J^i}(\mathcal{G})}$. We define a bijective map $\Sigma^{\mathcal{G}}_{J^i} : AP_{\mathcal{G}} \to AP_{\sigma^{\mathcal{G}}_{J^i}(\mathcal{G})}$ as shown in Table 6.2. The mapping $\Sigma^{\mathcal{G}}_{J^i}$ permutes any process indices appearing in an atomic proposition according to $\sigma^{\mathcal{G}}_{J^i}$ (abbreviated in the table to σ). We can show that $\rho^{\mathcal{G}}_{J^i}$, the path index map induced by $\sigma^{\mathcal{G}}_{J^i}$ is an isomorphism between adversaries in $Adv^{j}_{\mathcal{M}_{\mathcal{G}}}$ and adversaries in $Adv^{j'}_{\mathcal{M}_{\sigma^{\mathcal{G}}_{j}(\mathcal{G})}}$.

Lemma 6.2.16. *Let $\mathcal{G} \in \Gamma \setminus \Phi$. Let $J^i \in join^i(\mathcal{G})$ and let $(F^{J^i}_{\mathcal{G}}, J^i, \sigma^{\mathcal{G}}_{J^i})$ be a join reduction for J^i. For every adversary A of $\mathcal{M}_{\mathcal{G}}$, there exists some adversary A' of $M_{\sigma^{J^i}_{\mathcal{G}}(\mathcal{G})}$ such that A is isomorphic to A' under $\rho^{\mathcal{G}}_{J^i}$, the path index mapping induced by $\sigma^{\mathcal{G}}_{J^i}$.*

Proof. Let $\mathcal{G} \in \Gamma \setminus \Phi$ and let $J^i \in join^i(\mathcal{G})$ and let $(F^{J^i}_{\mathcal{G}}, J^i, \sigma^{\mathcal{G}}_{J^i})$ be a join reduction for J^i. Let $\mathcal{M}_{\mathcal{G}} = (S, s_0, Steps, Act, L)$ and $\mathcal{M}_{\sigma^{J^i}_{\mathcal{G}}(\mathcal{G})} = (S', s'_0, Steps', Act', L')$. Let A be an adversary of $\mathcal{M}_{\mathcal{G}}$ that is first-full with respect to J^i. Let α be a finite path in $Path_{fin}(s_0)$ and suppose $A(\alpha) = (a, \mu)$ for $(a, \mu) \in Steps(last(\alpha))$. Let $\alpha' = \rho^{\mathcal{G}}_{J^i}(\alpha)$ be the finite path in $Path_{fin}(s'_0)$ obtained under the path index map induced by $\sigma^{J^i}_{\mathcal{G}}$.

Suppose that $(a, \mu) = (r^k, \mu)$ corresponds to a transition associated with $process_k$ choosing a preference value of 0 or 1. Then $\mu(t_1) = \mu(t_2) = 0.5$ for $t_1, t_2 \in S$. If r^k is enabled from $last(\alpha)$ then since $last(\alpha') = \varsigma^{\mathcal{G}}_{J^i}(last(\alpha))$, $r^{\sigma^{J^i}_{\mathcal{G}}(k)}$ is enabled from $last(\alpha')$ (where $\varsigma^{\mathcal{G}}_{J^i}$ is the index map induced by $\sigma^{J^i}_{\mathcal{G}}$). Let $(a', \mu') = (r^{\sigma^{j}_{\mathcal{G}}(k)}, \mu')$ where $\mu'(t'_1) = \mu'(t'_2) = 0.5$ such that $t'_1, t'_2 \in S'$. It should be clear that $t'_1 = \varsigma^{\mathcal{G}}_{J^i}(t_1)$, $t'_2 = \varsigma^{\mathcal{G}}_{J^i}(t_2)$ and $\mu(t_1) = \mu(t'_1) = 0.5$ and $\mu(t_2) = \mu'(t'_2) = 0.5$. We can choose adversary $A_{\prime}$ such that $A_{\prime}(last(\alpha')) = (a', \mu')$.

We can consider each of the transitions shown in Figure 6.1 in a similar way. Thus we can construct $A\prime$ so that, from Definition 3.5.18, $A\prime$ is isomorphic to A under $\rho^{\mathcal{G}}_{J^i}$. $\square$

In Lemma 6.2.17 we show that, for every set of paths in the partition of an adversary of $\mathcal{M}_{\sigma^{\mathcal{G}}_{J^i}(\mathcal{G})[J^i]}$ there exists a randomised adversary of $\mathcal{M}_{\sigma^{\mathcal{G}}_{J^i}(\mathcal{G})[J^i]} \oplus F^{J^i}_{\mathcal{G}}$, that is stuttering equivalent with respect to the set of atomic propositions defined in Definition 6.2.11.

Lemma 6.2.17. *Let $\mathcal{G} \in \Gamma \setminus \Phi$, $J^i \in disjoint^i(\mathcal{G})$ and $\mathcal{R} = (F^{J^i}_{\mathcal{G}}, J^i, \sigma^{\mathcal{G}}_{J^i})$ be the join reduction for J^i. Let A be an adversary of $\mathcal{M}_{\mathcal{G}}$, and let A' be an adversary of $M_{\sigma^{J^i}_{\mathcal{G}}(\mathcal{G})}$ such that A is isomorphic to A' under $\rho^{\mathcal{G}}_{J^i}$, the path index mapping induced by $\sigma^{\mathcal{G}}_{J^i}$. Then there exists some randomised adversary E' of $\mathcal{M}_{\mathcal{R}(\mathcal{G})}$ such that $\rho^{\mathcal{G}}_{J^i}(P^A_{s_0}(J^i))$ and E' are stuttering equivalent w.r.t. $AP'_{\mathcal{R}(\mathcal{G})}$.*

Proof. Let $\mathcal{M}_{\sigma^{\mathcal{G}}_{J^i}(\mathcal{G})} = (S, s_0, Steps, Act, L)$ and $\mathcal{M}_{\mathcal{R}(\mathcal{G})} = (S', s'_0, Steps', Act', L')$. Let A be an adversary of $\mathcal{M}_{\mathcal{G}}$, and let A' be an adversary of $M_{\sigma^{J^i}_{\mathcal{G}}(\mathcal{G})}$ such that A is isomorphic to A' under $\rho^{\mathcal{G}}_{J^i}$, the path index mapping induced by $\sigma^{\mathcal{G}}_{J^i}$. Let $AP* \triangleq AP'_{\mathcal{R}(\mathcal{G})}$ and $\Pi = \rho^{\mathcal{G}}_{J^i}(P^A_{s_0}(J^i))$.

For a path, α, such that $\alpha \leq \gamma$ for $\gamma \in \Pi$ let,

$$\bar{\mathbf{P}}(\alpha) = \frac{\mathbf{P}(\alpha)}{Prob^{A'}_{s_0}(\Pi)}.$$

Furthermore if,

$$\alpha = s_0 \xrightarrow{a_1,\mu_1} s_1 \xrightarrow{a_2,\mu_2} \dots s_{n-1} \xrightarrow{a_n,\mu_n} s_n,$$

then for any $1 \leq i \leq n$, let $s_{i-1} \xrightarrow{a_i,\mu_i} s_i$ be a *first-round* transition under α if and only if a_i is an action local to $process_k$ and in every state in α up to s_{i-1}, $process_k$ has not yet received and sent a counter. Let the set of actions occurring in first-round transitions under α be denoted by $First(\alpha)$. Also, for $\gamma \in \Pi$ with, $\gamma = s_0 \xrightarrow{a_1,\mu_1} s_1 \xrightarrow{a_2,\mu_2} \dots s_{n-1} \xrightarrow{a_n,\mu_n} s_n \xrightarrow{a_{n+1},\mu_{n+1}} \dots$, let $second_{s_i}(\gamma)$ denote the finite path α starting in s_i with final state, s_{j+1} such that $s_j \xrightarrow{a_{j+1},\mu_{j+1}} s_{j+1}$ is not a first-round transition and every transition up to s_j is a first-round transition. Define,

$$Second_s(\Pi) = \{second_s(\gamma) | \gamma \in \Pi\}.$$

Let $H \subseteq Path_{fin}(s_0) \times Path_{fin}(s'_0)$ be the relation given by $\forall \alpha \in Path_{fin}(s_0)$, $\alpha' \in Path_{fin}(s'_0)$, $H(\alpha, \alpha')$ iff

$$trace^{AP*}(\alpha) \simeq trace^{AP*}(\alpha'),$$

and,

$$act^{Act^{\sigma^{\mathcal{G}}_{Ji}(\mathcal{G})}\setminus First(\alpha)}(\alpha) = act^{Act^{\mathcal{R}(\mathcal{G})}}(\alpha).$$

We define a randomised adversary, E', over $\mathcal{M}_{\mathcal{R}(\mathcal{G})}$ and sets, D_0, D_1, D_2,... such that $\forall n \geq 0$, $D_n \subseteq \{\alpha \leq \gamma | \gamma \in \Pi\}$, by induction. We show that for all $n \geq 0$,

IH1 For every $\alpha \in D_n$, $\alpha' \in cut^{E'}(n)$, if $H(\alpha, \alpha')$ then for every $m < n$ there exists prefixes $\beta \leq \alpha$ and $\beta' \leq \alpha'$ such that $\beta \in D_m$, $\beta' \in cut^{E'}(m)$ and $H(\beta, \beta')$.

IH2 If μ_n, μ'_n are the distributions over D_n and $cut^{E'}(n)$, respectively, defined by, for $\alpha \in D_n$, $\alpha' \in cut^{E'}(n)$, $\mu_n(\alpha) = \bar{\mathbf{P}}(\alpha)$ and $\mu'_n(\alpha') = Prob^{E'}_{s_0}(\mathcal{C}(\alpha'))$ then $\mu_n \sqsubseteq_H \mu'_n$.

Base case: For $n = 0$, $cut^{E'}(0) = \{s'_0\}$. Let $D_0 = \{s_0\}$. Immediately, IH1 holds. By definition $\bar{\mathbf{P}}(s_0) = Prob^{E'}_{s'_0}(\mathcal{C}(s'_0)) = 1$. Therefore, $\mu'_0 \sqsubseteq_H \mu_0$ and so IH2 holds.

Induction step: Assume that IH1 and IH2 hold for some $n \geq 0$. We define D_{n+1} and $E'(\alpha')$ for all $\alpha' \in cut^{A'}(n)$.

Suppose $D_n = \{\alpha_0, \alpha_1, \ldots, \alpha_{k_n}\}$. For $0 \leq j \leq k_n$, let $s_j = last(\alpha_j)$. Define $D_{n+1} = \cup_{j=0}^{k_n}\{\alpha_j.\beta | \beta \in Second_{s_j}(\Pi)\}$.

Suppose that $\alpha = \alpha_j.\beta \in D_{n+1}$. Then $\beta = \gamma \xrightarrow{a,\mu} t$ such that $last(\gamma) \xrightarrow{a,\mu} t$ is not a first-round transition under α while every transition in γ is.

Suppose, for $\alpha' \in cut^{E'}(n)$, $H(\alpha_j, \alpha')$. Therefore,

$$act^{Act^{\sigma^{\mathcal{G}}_{Ji}(\mathcal{G})}\setminus First(\alpha)}(\alpha_j) = act^{Act^{\mathcal{R}(\mathcal{G}}}(\alpha').$$

Since the transitions in γ are all first-round transitions under α and $last(\gamma) \xrightarrow{a,\mu} t$ is not a first-round transition, it should be clear that a is enabled from $last(\alpha')$. Let

$$E'(\alpha')(a, \mu') = \frac{\mathbf{P}(\gamma)}{\sum_{\beta \in Second_{s_j}}(\Pi)}.$$

By the definition of D_{n+1} and $cut^{E'}(n+1)$, IH1 and IH2 are satisfied. From Lemma 3.5.13 it follows that $\Pi \simeq A'$.

□

It should be apparent from Lemmas 6.2.16 and 6.2.17 and the definition of the path index map that,

Lemma 6.2.18. *For every adversary A in $\mathcal{M}_{\mathcal{G}}$ ($\mathcal{G} \in \Gamma \setminus \Phi$) there exists a join reduction $\mathcal{R} = (F_{\mathcal{G}}^{J^i}, J^i, \sigma_{J^i}^{\mathcal{G}})$ such that $P_{s_0}^{A}(J^i)$ is isomorphic to $P_{s_0'}^{A'}(\sigma_{\mathcal{G}}^{J^i}(J^i))(= \rho_{J^i}^{\mathcal{G}}(P_{s_0}^{A}(J^i)))$ for some adversary A' in $\mathcal{M}_{\sigma_{J^i}^{\mathcal{G}}(\mathcal{G})}$ with $P_{s_0'}^{A'}(\sigma_{\mathcal{G}}^{J^i}(J^i))$ stuttering equivalent to some randomised adversary E' in $\mathcal{M}_{\mathcal{R}(\mathcal{G})}$ w.r.t. $AP'_{\mathcal{R}(\mathcal{G})}$.*

6.2.5 Proof of Theorem 6.2.1

Proof. From Lemma 6.2.10, the set Γ is reducible to $\{\mathcal{M}_{\mathcal{G}(2)}\}$ with probability 1. By Lemmas 6.2.12, 6.2.13 and 6.2.18, the conditions of Definition 6.1.6 are satisfied by $\mathcal{M}_{\Gamma}$ (under the join reductions). Thus, $\mathcal{M}_{\Gamma}$ is probabilistic degenerative with base Φ. By Lemma 6.2.2 *Property 1* holds for $\mathcal{M}_2$. Since *Property 1* is unindexed with appropriate atomic propositions, by Theorem 6.1.7, it is satisfied by $\mathcal{M}_{\mathcal{G}}$ for all $\mathcal{G} \in \Gamma$. $\square$

6.3 Discussion

6.3.1 Specifying Degenerative Systems

Clearly the applicability of our approach is strongly dependent on the form of the model used to represent a system: a parameterised family of models of a system that appears to degenerate may not be degenerative. Therefore, it would be useful to identify some heuristics that can be used to guide the manner in which the system is modelled. For example, it is clear that in order for a family of models to be degenerative then for a process there must be a reachable local state that once reached guarantees that the process will 'acquiesce' (for example by terminating or by only passing on received messages). It must also be the case that with probability 1 some process (or set of processes) will reach this state. Furthermore, we must have that, for any execution along which a process or set of processes reach a quiescent state, any action performed by such a process must be a stutter action.

Note that these heuristics do not guarantee that a family of models is degenerative. In fact, much of the work in establishing this is based on identifying the sets of 'reduced' variables and domains. In terms of the IRP, for example, it was necessary to consider a very small subset of the variables and their domains. This was due to the cyclic behaviour of each process in the protocol. In the TIP example a process did not return to a previously visited state and therefore it was only necessary to ensure that actions of the degenerating processes were stuttering actions. For IRP, on the other hand, it was necessary to ensure that every action that occurred before a degenerate state was reached was a stutter action. This can therefore be quite a severe restriction (but note that in practice this need not limit the properties in which we are interested, as was the case for the IRP).

6.3.2 Convergent Properties

In terms of work closely related to our inductive schema for degenerative systems we mention the work by Duflot et al. described in [27] and [28]. In [27], an approach to proving the convergence of self-stabilising randomised protocols for parameterised rings is given. By appealing to results of Markov theory it is shown that, given a non-increasing measure on the state space of the model, if there exists a 'distance' measure between states and an ordering relation on the measure that satisfies certain conditions then it is possible to deduce that the protocol will converge to some *legitimate* set of states with probability 1. In [28] this technique is applies to an adaptation of Lehmann-Rabins solution to the dining philosophers problem that does not require fairness constraints, and show that for the modified algorithm a philosopher will eventually eat with probability 1.

The technique is described only in terms of rings with synchronous communication, although it appears possible that it could be extended to other topologies and perhaps also for asynchronous communication. The technique is also limited to proving 'convergence' to a set of states with probability 1 and therefore temporal logic is not considered. Our technique, on the other hand, provides a theoretical framework to prove quantitative properties of systems expressed as (a restricted class of) $\mathrm{LTL}_{\backslash\mathcal{X}}$ formula. However, it may be that the type of properties that are of interest that we can actually analyse by this method are in practice only converging probability 1 properties: the two properties that we consider for the TIP and IR examples are of this form. This would require further rigorous study to determine.

As with our technique, the approach of Duflot et al. is not automated and in particular requires the identification of a measure on the set of states along with an ordering. Similarly our technique requires reductions to be defined and the identification of sets of reduced variables. One of the benefits of the convergence technique is that the conditions on the measure and the ordering are defined locally i.e. it is only necessary to consider single transitions of the probabilistic model rather than paths. This is a weakness of our technique that is inherent to establishing stutter equivalence. It might be that we can consider other relations, such as the weak simulation relations described in [58], that are defined 'locally', i.e. with respect to individual transitions. Note, however, that these are defined in terms of MDPs rather than in terms of adversaries, and we would therefore need to modify our approach to take this into account. Using a relation of this type would have the added benefit of allowing us to consider PCTL, rather than LTL properties. Under a simulation style relation we can only establish properties of the form $\mathcal{P}_{\geq p}[\phi]$ (where ϕ is a PCTL path formula).

Assuming that the convergence approach of [27] and [28] can be extended to prove convergence properties of protocols arranged in arbitrary topologies, it seems likely that this technique would also be applicable to degenerative systems. In fact, it might be particularly useful in showing that a set of topologies is reducible to some base with probability 1. In the case of the TIP, for example, we can define a distance measure to be the number of level 1 nodes and an ordering $<$, the usual ordering on positive natural numbers and for Itai Rodeh we could use the same order and define a distance measure to be the size of the ring. It remains to be determined whether these satisfy the definitions of [27] and [28].

On the other hand, it might be also true that our technique could be extended to verification of, for example, self-stabilising protocols. Consider, for example, the self-stabilising protocol of [34]. Although this does not degenerate in the manner we have defined, it seems apparent that for a system with $N > 1$ processes, once a configuration has been reached in which there are m, say, tokens then this system is related to the system with m processes and m tokens. The main difficulty in showing this would be in determining a relation of an appropriate form such that we could generalise this principle to any system of this type.

One of the weaknesses of our approach as it stands is that we consider families of models derived from PRISM specifications. PRISM provides support for verification of PCTL properties, whereas we establish results for a class of quantitative LTL properties. Although we can still verify a subset of properties that are in both LTL and PCTL, this is a severe restriction. One solution to this, described above, would be to adapt our results to consider a relation other than stuttering equivalence. Alternatively, our approach could be adapted to consider families of models specified in other modelling languages, such as Probmela, which is supported by the Liquor model checking tool. Liquor can be used to verify both LTL and PCTL properties and therefore may be a more appropriate verification tool in this instance.

Summary We have presented a definition of a probabilistically degenerative family of models parameterised by a communication topology. Based on this definition we have established that in order to verify certain types of property for a probabilistically degenerative family of models, it is sufficient to consider some subset of the family that corresponds to a base subset of the communication topology. This is demonstrated by considering a family of MDP models of the Itai-Rodeh leader election protocol, which we show is probabilistically degenerative.

Chapter 7

Probabilistic Parameterised Verification of Semi-Degenerative Systems

Outline We consider randomised distributed systems that are degenerative but that cannot be considered as deterministically or probabilistically degenerative. We give a definition for a family of probabilistic models to be semi-degenerative. A simple gossip protocol is provided as an example of a semi-degenerative family of models and the degenerative nature of the protocol is exploited in order to establish that given PCTL properties that hold for a base system model hold for the entire family.

7.1 Introduction

In Chapters 5 and 6 we presented a technique for verification of degenerative systems by induction, establishing that, for a degenerative family of models, if a property is true for some base family of models then it is true for the entire family of models. We considered two example systems in which, for each model the system was guaranteed to reach a state that corresponded to a state in a 'smaller' model. The definition of deterministically and probabilistically degenerative families of MDP models from these chapters are such that the models must 'converge' to a base system model with probability 1. We consider now families of models that degenerate, but that may also *terminate*. Thus, with probability 1 a model reaches a degenerate state *or* it reaches a *terminal* state from which it is not possible reach a state in a smaller model. It is possible that we could extend the approach described in the previous two chapters to be applicable to these systems. However, we consider here an illustration of an alternative approach through analysis of a simple gossip protocol example.

7.2 Semi-Degenerative Families of Models

Like in the previous chapters we give the definition for a family of models to be degenerative by considering a structural relation between models in the family. We consider here a relation between models where one is 'embedded' in the other. Whereas in the previous chapters we considered MDPs, here we consider DTMC models. We discuss the possibility of extending this work to MDP models at the end of this chapter.

Definition 7.2.1. *Let $\mathcal{D} = (S, s_0, \mathbf{P}, L)$ be a DTMC. A DTMC $D' = (S', s'_0, \mathbf{P}', L')$ is a sub-chain of D (denoted $D' \sqsubseteq D$) if and only if $S' \subseteq S$, $s' \in S' \Leftrightarrow s'$ is reachable from s'_0, and for all $s', t' \in S'$, $\mathbf{P}'(s', t') = \mathbf{P}(s', t')$.*

Based on Definition 7.2.1 we can give a definition for a family of models (over a set of specifications) to be *semi-degenerative*. A family of DTMC models is *semi-degenerative* if every 'smaller' model is a sub-chain of a 'larger' model (we formalise this below). The use of the sub-chain relation in our definition is somewhat different from that of the stuttering equivalence relation employed for the deterministically and probabilistically degenerative families of models of the previous two chapters. Intuitively, this relation implies that the behaviour of some model, before it reaches some state of a 'smaller' model, is independent of the behaviour of the smaller model, but thereafter it is identical to that of the smaller model. Under the stuttering equivalence relation, on the other hand, the behaviour of a model before and after a state of a smaller model is reached, need only be equivalent to the behaviour of the smaller model up to stuttering.

In this chapter we assume that the topology of the system is regular and therefore we can consider the family of specifications to be parameterised by the size of the system (number of processes) rather than by the communication topology.

Definition 7.2.2. *For some $c \geq 2$, let $\mathcal{S} = \{\mathcal{S}(N) | N \geq 1\}$ be a family of specifications for a system of size N and let $\mathbf{D} = \{\mathcal{D}(\mathcal{S}(N)) | N \geq 1\}$ be the corresponding family of DTMC models. $\mathbf{D}$ is semi-degenerative with base c if and only if for all $N > c$, $\mathcal{D}(\mathcal{S}(N-1)) \sqsubseteq \mathcal{D}(\mathcal{S}(N))$.*

We describe a family of models satisfying the above conditions as *semi*-degenerative because it is not necessarily possible to reach the 'base' system model $\mathcal{D}(\mathcal{S}(c))$ from every state. In other words there may exist states from which the probability of reaching a state in $\mathcal{D}(\mathcal{S}(c))$ is zero. This highlights another difference between the semi-degenerative systems and the those types of degenerative behaviour given in the previous chapters, where we assumed that we were guaranteed to reach a 'base' system model. In Definition 7.2.3 we distinguish between three types of states of a DTMC in terms of some sub-chain: terminal, degenerative and semi-degenerative. For a DTMC, $\mathcal{D} = (S, s_0, \mathbf{P}, L)$, a set $T \subseteq S$, and a state $s \in S$, let $p_s(T) = Prob_s(\{\omega \in Path(s) | \omega = s_0 \to \ldots \to t \to \ldots \text{ for some } t \in T\})$.

Definition 7.2.3. *Suppose $\mathcal{D} = (S, s_0, \mathbf{P}, L)$ is a DTMC and $D' = (S', s_0', \mathbf{P}', L')$ is a sub-chain of D. Then, for $s \in S$, s is terminal w.r.t. S' if and only $p_s(S') = 0$, s is degenerative w.r.t. S' if and only if $p_s(S') = 1$, and s is semi-degenerative w.r.t. S' otherwise.*

We prove reachability properties for a family of semi-degenerative models. As discussed above the behaviour of a model prior to reaching a state of a sub-chain is independent of the sub-chain's behaviour. In order to prove properties of a family of semi-degenerative models we need to establish some results for the behaviour of each model in the family with respect to the property. This is strictly dependent on the property and model under consideration, thus it is difficult to provide a general theorem for proving results of semi-degenerative families of models. In the following section we prove two example properties, by induction, for an example semi-degenerative system.

7.3 Case Study: A Gossip Protocol

7.3.1 Gossip Protocols

We consider a class of randomised, distributed algorithms commonly known as *gossip* or *epidemic protocols*, which disseminate information in a peer-to-peer network. The ideas behind gossip protocols are originally derived from the field of epidemiology, the study of disease spread.

The *SIR* model of disease spread (see for example [11]) subdivides a population into those that are susceptible (have not yet contracted a disease), those that are infective (have contracted the disease and may pass it on) and those that are removed (had the disease and are now immune or dead). In the simplest form of the model, susceptible individuals become infective (with some probability, B, say) and infective individuals then become removed (again with some associated probability, Q say). It is also possible to extend this model to consider births, natural deaths, vaccination, spatial spread, mutation of the disease *etc.* but only the basic model will be considered here. It is usual to employ a system of differential equations to model the spread of disease in a population. This deterministic approach is an accurate representation for large populations, but for smaller populations it is more appropriate to use stochastic methods.

Gossip protocols are designed to propagate information throughout a network of sites, in much the same way as diseases. An 'infection' will appear at some site and the information that this represents must be propagated throughout the network. This is done by randomly choosing a (set of) site(s) to 'infect'. Each 'infected' site may then eventually become 'removed', according to some mechanism, and stop transmitting the information. Whereas the study of epidemiology is concerned with the containment of a disease, gossip protocols are designed to rapidly propagate information to as many sites as possible.

In spite of their advantages, gossip protocols are not suitable in every situation and have two significant weaknesses. Firstly, because the protocol is based on randomised behaviour, there is no guarantee that everyone will receive an 'infection'. This problem can be alleviated by infrequently executing a more heavyweight protocol, that gives a deterministic guarantee on the spread of the information, in parallel with the gossip protocol.

However, this solution conflicts with the second weakness, which is that, although the behaviour of the protocol at a local level is relatively simple, the precise nature of the global behaviour is often difficult to understand. This means that the use of gossip protocols in conjunction with other protocols can be unpredictable. The analysis of these systems is therefore very important. Most of the investigations into the behaviour of these protocols so far has relied on manual proof, experimentation and simulation. However, by making use of probabilistic model checking, for systems of a fixed size, the protocols can be verified automatically.

In the following sections we consider a simple gossip-style protocol and discuss how the PRISM probabilistic model checking tool is employed to verify it. We then tackle the PMCP based on the *degenerative* nature of the protocol using the approach described in the previous section.

7.4 SIR Protocol

We consider a distributed peer-to-peer network of sites with a fully connected topology. At the start of the protocol one site will be considered infective, i.e. it has some information for distribution to the remaining susceptible sites. Any infective site can then infect any susceptible site, with the susceptible site choosing to become infective with some probability, β. Any infective site may also, at any point, choose to become removed with probability ρ, in which case that site is no longer interested in forwarding information and no longer participates in the protocol.

7.4.1 Model Checking the SIR Protocol with PRISM

We make some simplifying assumptions for the protocol. The first, is that there is only ever one form of information present in the system at any one time. Second, we consider our system to be free from failure, a common assumption when performing model checking.

The PRISM specification for the SIR protocol with three network sites is given in Appendix G. Notice that the protocol is modelled as a DTMC. For a system of size N, we consider each network site as a process and model a single site as a PRISM module $site_i$ $(1 \leq i \leq N)$. Each module has a single local variable, $state_i$ $(0 \leq i \leq N-1)$, tracking the state of the process. There are three possible states: *Susceptible*, $Infective$, *Removed.* Any process that is in the *Susceptible* state randomly chooses an $Infective$ process (assuming that there is one) and will then become $Infective$ with probability B, where B is a global constant in the range $[0 \ldots 1]$. An $Infective$ process is only able to move to the *Removed* state, which it does so with probability Q, where Q is also a global constant in the range $[0 \ldots 1]$. Note that initially one module is in the $Infective$ state while the remainder are initialised to be *Susceptible*.

There are a number of properties that are of interest to us when verifying our model. Here we concentrate on two properties, which we respectively define informally and formally (in PCTL) below.

Property I1 With probability 1, the spread of the update will terminate.

$$\mathcal{P}_{\geq 1} \qquad [\Diamond(\mathit{state}_1! = Infective\ \&\ \mathit{state}_2! = Infective\ \&...\&\ \mathit{state}_N! = Infective)]$$

Property I2 Upon the spread of the update terminating, with probability greater than or equal to a half, there will be no susceptibles remaining.

$$\mathcal{P}_{\geq 1/2} \qquad [\Diamond(\mathit{state}_1! = Infective\ \&\ ...\&\ \mathit{state}_N! = Infective\ \&$$
$$!(\mathit{state}_1 = \mathit{Susceptible} \mid \mathit{state}_2 = \mathit{Susceptible} \mid ... \mid \mathit{state}_N = \mathit{Susceptible}))]$$

These properties have been verified using PRISM for $N = 2, 3, ..., 15$ sites.

A Population Model of the SIR Protocol

The model in Appendix E is an individuals model. Our model is highly symmetric and our properties are not dependent on the identity of any particular site. We can therefore employ a 'population' level model, with fewer states where, rather than maintaining the state for each process in a system, we instead count the number of processes in each state.

A PRISM specification of the population model is given in Appendix H. We model the protocol with a single module that has two state variables, s and i, which maintain the number of susceptible and infective sites in the protocol. Note that we omit the number of removed sites since this can be calculated from the total number of sites and the number of susceptible and infective sites. If a susceptible site becomes infective then s is decremented by one and i incremented accordingly. If an infective site becomes removed then i is decremented by one. The only other possible transition is a self-loop in which case s and i are unchanged.

Consider the probabilities of a site changing its state. Let s and i denote the number of susceptibles and infectives respectively. Let $\mathbf{P}[S \to I]$ denote the probability of a susceptible site becoming infective. Let $\mathbf{P}[I \to R]$ denote the probability of an infective site becoming removed and let $\mathbf{P}[S \to S]$ denote the probability of remaining in the same state. Then:

$$\mathbf{P}[S \to I] = \frac{i}{s.i + i}.B.s = B.\frac{s}{s+1}$$
$$\mathbf{P}[I \to R] = \frac{1}{s.i + i}.Q.i = Q.\frac{1}{s+1}$$

$$\begin{aligned}\mathbf{P}[S \to S] &= 1 - (\mathbf{P}[S \to I] + \mathbf{P}[I \to R]) \\ &= 1 - (B.\frac{s}{s+1} + Q.\frac{1}{s+1}) \\ &= 1 - \frac{B.s + Q}{s+1}\end{aligned}$$

Note that we also introduce a global positive integer constant, N, the total number of sites, to our population specification. Rather than defining separate specifications for each size of system as for the individual specification, with the population specification we can define a single specification using the parameter N to represent the number of sites.

We can verify the three properties specified below for at least one hundred sites, greater than the number of sites for which we could verify the individual level SIR protocol. Note the comparability of *Property P1* and *Property P2* to *Property I1* and *Property I2* of the individual level models given above.

Property P1 With probability 1, the spread of the update will terminate.

$$\mathcal{P}_{\geq 1}[\Diamond(i = 0)]$$

Property P2 With probability at least a half, when the spread of the infection terminates, there will be no susceptibles remaining.

$$\mathcal{P}_{\geq 1/2}[\Diamond(i = 0 \& s = 0)]$$

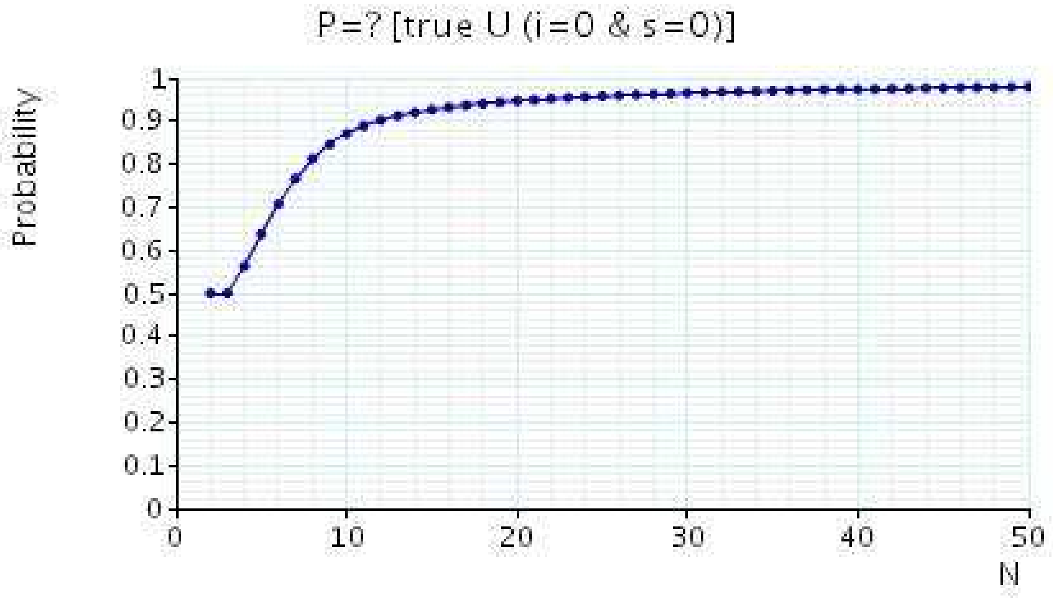


Figure 7.1: Probability of *Property P2* holding against N

Property P3 With probability at least a half, eventually the number of infectives will be greater than the number of susceptibles.

$$P_{\geq 1/2}[\Diamond(i > s)]$$

As discussed in Section 3.4, PRISM can return the actual probability of a property holding, as well as simply checking a bound. This probability can be plotted against a varying parameter, as can be seen in Figure 7.1, where the total number of sites (N) is plotted against the probability of the update spread completing with no remaining susceptible sites, for $N = 2, 3, \ldots 50$.

7.5 Parameterised Model Checking of the SIR Protocol

The SIR protocol is a degenerative system. Intuitively, for any network of N sites, once a site becomes removed, the protocol behaves as a system of $N - 1$ sites. However, notice that upon removing a site we may no longer have any infective sites and the protocol will terminate, possibly with some susceptible sites remaining. Figure 7.2 shows the DTMCs generated from the population level model for two, three, four and five sites (note that each state is shown as x, y, where $s = x$ and $i = y$). It should be apparent that the

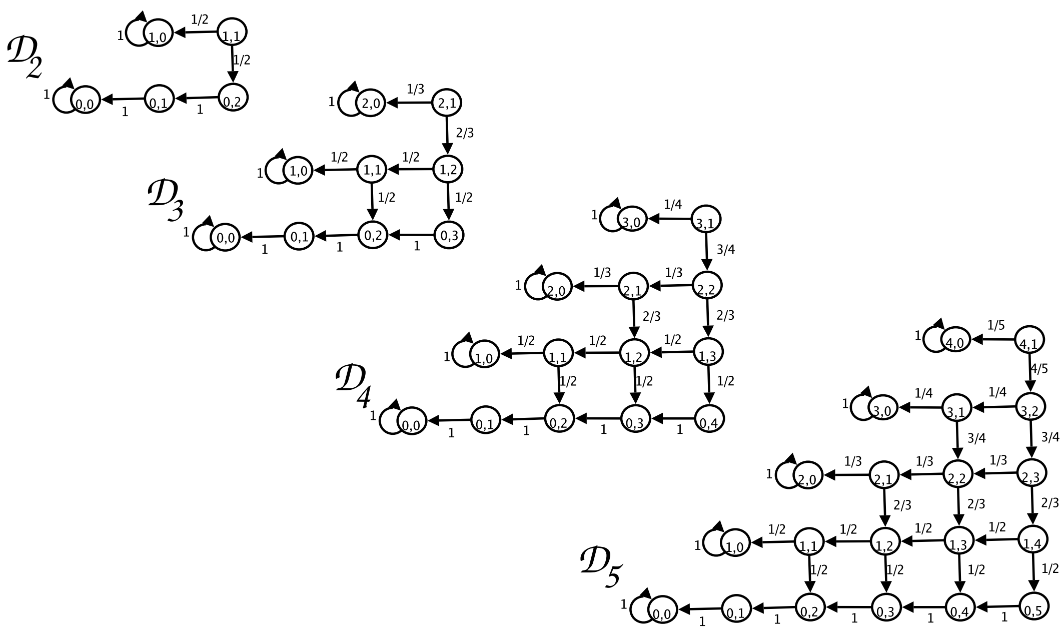


Figure 7.2: DTMCs for the SIR protocol for 2,3,4 and 5 sites

smaller models are subgraphs of the larger ones. Note also, however, that there exist states where the protocol terminates without reaching a state of the smaller model $\mathcal{D}_2$. For example, state $(4, 0)$ is a *terminal* state. Hence, it would appear that the protocol is *semi-degenerative* in the manner defined in Section 7.2.

In the sequel, for $n \geq 2$, let $\mathcal{S}(n)$ denote the population SIR protocol PRISM specification with $N = n$. Let $\mathcal{D}_n = \mathcal{D}(\mathcal{S}(n))$, $\mathcal{S} = \{\mathcal{S}(n)|n \geq 2\}$ and $\mathbf{D} = \{\mathcal{D}_n|n \geq 2\}$.

Lemma 7.5.1. *The family of models* $\mathbf{D}$ *is semi-degenerative with base 3.*

Proof. For $N \geq 3$ let $\mathcal{D}_N = (S_N, s_0^N, \mathbf{P}_N, L_N)$. Let $N > 3$. Clearly $S_{N-1} \subseteq S_N$. Suppose that $s \in S_{N-1}$. Then $s = (x, y)$ such that $x + y \leq N - 1$. It should be clear that s is reachable from s_0^{N-1}. If $s = (x, y)$ is reachable from $s_0^{N-1} = (N - 2, 1)$ then $s \in S_{N-1}$.

Since $\mathcal{S}(N)$ is identical to $\mathcal{S}(N-1)$ except for the initialisation of parameter N and since $\mathcal{S}(N)$ is independent of N, $\forall s, s' \in S_{N-1}$, $\mathbf{P}_N(s, s') = \mathbf{P}_{N-1}(s, s')$. ☐

By employing Lemma 7.5.1 we now prove that *Property P2* and *Property P3* are satisfied by $\mathcal{D}_N$ for all $N \geq 3$.

Property P2

We show that the probability of the SIR protocol terminating with no remaining susceptibles is at least a half for any size of system. We verify this property in terms of the population level specification of the SIR protocol.

Theorem 7.5.2. *If ϕ denotes Property P2 then for any $n \geq 2$, $\mathcal{D}_n \models \phi$.*

Proof. By induction on N, the number of processes.

As should be apparent from the DTMCs given in Figure 7.2, from the initial state in a model $\mathcal{S}_{N+1}$, it is possible to reach a state in $\mathcal{S}_N$ that is not the initial state. In fact, any path from the initial state in $\mathcal{S}_{N+1}$ must pass through a state, (s, i), in $\mathcal{S}_N$ such that $s + i = N$. Therefore, it is not enough to prove our property holds in the initial state of every model. We must also show that it holds in any state where $s + i = N$.

Base Case:

For $N = 3$, we have verified by model checking in PRISM that,

$$\forall r = (s, i) \text{ s.t. } s + i = 3,\ \mathcal{S}_3, r \models \mathcal{P}_{\geq 1/2}[\Diamond(s = 0 \& i = 0)].$$

Induction Step:

For $N \geq 3$ assume that

$$\forall r = (s, i) \text{ s.t. } s + i = N,\ \mathcal{S}_N, r \models \mathcal{P}_{\geq 1/2}[\Diamond(s = 0 \& i = 0)]$$

Let $p_{x,y}$ denote the probability of reaching state $(s = 0, i = 0)$ from the state $(s = x, i = y)$. Then, for $0 \leq m \leq N - 1$, we can write the probability of reaching $(s = 0, i = 0)$ from any state with $s + i = N + 1$ as the sum of the probability of states with $s + i = N$.

$$\begin{aligned}
p_{m,N-m+1} &= 1.\frac{1}{2}.\frac{2}{3}.\frac{3}{4}.\frac{4}{5}\cdots\frac{m-1}{m}.\frac{m}{m+1}.p_{0,N}\\
&+ \frac{1}{2}.\frac{2}{3}.\frac{3}{4}.\frac{4}{5}\cdots\frac{m-1}{m}.\frac{m}{m+1}.p_{1,N-1}\\
&+ \frac{1}{3}.\frac{3}{4}.\frac{4}{5}\cdots\frac{m-1}{m}.\frac{m}{m+1}.p_{2,N-2}\\
&+ \frac{1}{4}.\frac{4}{5}\cdots\frac{m-1}{m}.\frac{m}{m+1}.p_{3,N-3}\\
&\vdots\\
&+ \frac{1}{m}.\frac{m}{m+1}.p_{m-1,N-m+1}\\
&+ \frac{1}{m+1}.p_{m,N-m}
\end{aligned}$$

Notice that, with the exception of the final term, the denominators and numerators in each of the products all cancel, hence,

$$\begin{aligned}
p_{m,N-m+1} &= \frac{1}{m+1}\sum_{k=0}^{m} p_{k,N-k}\\
&= \frac{1}{m+1}\left(1+\sum_{k=1}^{m} p_{k,N-k}\right) \text{ since, } \forall y \geq 0, p_{0,y} = 1\\
&\geq \frac{1}{m+1}\left(1+\sum_{k=1}^{m}\frac{1}{2}\right) \text{ from our inductive assumption}\\
&= \frac{1}{m+1}\left(1+\frac{1}{2}m\right)\\
&= \frac{m+2}{m+1}.\frac{1}{2}\\
&\geq \frac{1}{2} \text{ since } \frac{m+2}{m+1} \geq 1
\end{aligned}$$

Finally, notice that the behaviour of the initial state (i.e. when $m = N$) is actually somewhat different from the other states since removing an infective results in a transition into the state $(s = N-1, i = 0)$, and note that $p_{N-1,0} = 0$ for any N. Therefore we need to resolve this case separately as follows:

$$\begin{aligned}
p_{N,1} &= \frac{N}{N+1}.p_{N-1,2}\\
&\geq \frac{N}{N+1}.\frac{N+1}{N}.\frac{1}{2}\\
&= \frac{1}{2}
\end{aligned}$$

Therefore, by induction,

$$\forall N \geq 3, \forall r = (s, i) \text{ s.t. } s + i = N \; \mathcal{S}_N, r \models \mathcal{P}_{\geq 1/2}[\Diamond(s = 0 \& i = 0)].$$

Since, for $s = N - 1, i = 1, \; s + i = N$,

$$\forall N \geq 3, \; \mathcal{S}_N \models \mathcal{P}_{\geq 1/2}[\Diamond(s = 0 \& i = 0)]$$

□

Property P3

Notice that the set of states satisfied by $(i > s)$ is not a subset of $\mathcal{S}_2$ as for *Property P2*. Nonetheless our proof follows similar lines to that for *Property P2*.

Theorem 7.5.3. *For $N \geq 2, \mathcal{S}_N \models \mathcal{P}_{\geq 1/2}[\Diamond(i > s)]$.*

Proof. By induction on N.

Base Case:

For $N = 2$, we have verified by model checking in PRISM that,

$$\forall r = (s, i) \text{ s.t. } s + i = 3, \; \mathcal{S}_3, r \models \mathcal{P}_{\geq 1/2}[\Diamond(i > s)].$$

Induction Step:

For $N \geq 2$ assume that

$$\forall r = (s, i) \text{ s.t. } s + i = N, \; \mathcal{S}_N, r \models \mathcal{P}_{\geq 1/2}[\Diamond(i > s)]$$

Let $p_{x,y}$ denote the probability of reaching state $(s = 0, i = 0)$ from the state $(s = x, i = y)$.

Note that for $s + i = N + 1$, $p_{s,i} = 1$ if $i > s$. In particular, this is true if s is less than or equal to $(N + 1)/2$ for N odd and less than or equal to $N/2$ for N even.

If N is odd then we have that, for $(N+1)/2 \leq m < N$,

$$\begin{aligned}
p_{m,N-m+1} &= \frac{\frac{N+1}{2}}{\frac{N+1}{2}+1}.\frac{\frac{N+1}{2}+1}{\frac{N+1}{2}+2}\cdots\frac{m-1}{m}.\frac{m}{m+1}.p_{(N+1)/2,(N+1)/2} \\
&+ \frac{1}{\frac{N+1}{2}+1}.\frac{\frac{N+1}{2}+1}{\frac{N+1}{2}+2}\cdots\frac{m-1}{m}.\frac{m}{m+1}.p_{(N+1)/2,(N+1)/2-1} \\
&\vdots \\
&+ \frac{1}{m}.\frac{m}{m+1}.\frac{m-1}{m}.p_{m-2,N-m+2} \\
&+ \frac{1}{m}.\frac{m}{m+1}.p_{m-1,N-m+1} \\
&+ \frac{1}{m+1}.p_{m,N-m}
\end{aligned}$$

Notice that, with the exception of the final term, the denominators and numerators in each of the products all cancel, hence,

$$\begin{aligned}
p_{m,N-m+1} &= \frac{N+1}{2}.\frac{1}{m+1}.1 \\
&+ \frac{1}{m+1}.p_{(N+1)/2,(N+1)/2-1} \\
&\vdots \\
&+ \frac{1}{m+1}.p_{m-2,N-m+2} \\
&+ \frac{1}{m+1}.p_{m-1,N-m+1} \\
&+ \frac{1}{m+1}.p_{m,N-m}
\end{aligned}$$

Therefore,

$$\begin{aligned}
p_{m,N-m+1} &= \frac{1}{m+1}\left(\frac{N+1}{2} + \sum_{k=0}^{m-(N+1)/2} p_{m-k,N-m+k}\right) \\
&\geq \frac{1}{m+1}\left(\frac{N+1}{2} + \sum_{k=1}^{m-(N+1)/2} \frac{1}{2}\right) \text{ from our inductive assumption} \\
&= \frac{1}{m+1}\left(\frac{N+1}{2} + (m - \frac{N+1}{2} + 1)\frac{1}{2}m\right) \\
&= \frac{(N+3)/2+m}{m+1}.\frac{1}{2}
\end{aligned}$$

Thus,

$$\begin{aligned} & p_{m,N-m+1} \geq \frac{1}{2} \\ \iff & \frac{(N+3)/2+m}{m+1} \geq 1 \\ \iff & N+3 \geq 2 \\ \iff & N \geq -1 \end{aligned}$$

We have that $N \geq 2$ and thus $p_{m,N-m+1} \geq \frac{1}{2}$.

Finally, notice that the behaviour of the initial state (i.e. when $m = N$) is actually somewhat different from the other states since removing an infective results in a transition into the state $(s = N-1, i = 0)$, and note that $p_{N-1,0} = 0$ for any N. Therefore we need to resolve this case separately as follows:

$$\begin{aligned} p_{N,1} &= \frac{N}{N+1}.p_{N-1,2} \\ &\geq \frac{N}{N+1}.\frac{1}{N}.\left(\frac{N+1}{2} + \left(N-1-\frac{N+1}{2}+1\right)\frac{1}{2}\right) \\ &= \frac{1}{N+1}.\frac{1}{2}.\left(2N+1-\frac{N+1}{2}\right) \\ &= \frac{1}{N+1}.\frac{1}{2}.\left(\frac{3N+1}{2}\right) \\ &= \frac{1}{2}\frac{3N+1}{2N+2} \end{aligned}$$

Thus,

$$\begin{aligned} & p_{N,1} \geq \frac{1}{2} \\ \iff & \frac{3N+1}{2N+2} \geq 1 \\ \iff & 3N+1 \geq 2N+2 \\ \iff & N \geq 1 \end{aligned}$$

Since $N \geq 2$, $p_{N,1} \geq \frac{1}{2}$.

Similarly, if N is even, for $N/2 \leq m < N$,

$$\begin{aligned}
p_{m,N-m+1} &= \frac{\frac{N}{2}+1}{\frac{N}{2}+2}.\frac{\frac{N}{2}+2}{\frac{N}{2}+3}\cdots\frac{m-1}{m}.\frac{m}{m+1}.p_{N/2,N/2+1} \\
&+ \frac{1}{\frac{N}{2}+2}.\frac{\frac{N}{2}+2}{\frac{N}{2}+3}\cdots\frac{m-1}{m}.\frac{m}{m+1}.p_{N/2+1,N/2-1} \\
&\vdots \\
&+ \frac{1}{m}.\frac{m}{m+1}.\frac{m-1}{m}.p_{m-2,N-m+2} \\
&+ \frac{1}{m}.\frac{m}{m+1}.p_{m-1,N-m+1} \\
&+ \frac{1}{m+1}.p_{m,N-m}
\end{aligned}$$

Notice that, with the exception of the final term, the denominators and numerators in each of the products all cancel, hence,

$$\begin{aligned}
p_{m,N-m+1} &= \left(\frac{N}{2}+1\right).\frac{1}{m+1}.1 \\
&+ \frac{1}{m+1}.p_{N/2+1,N/2-1} \\
&\vdots \\
&+ \frac{1}{m+1}.p_{m-2,N-m+2} \\
&+ \frac{1}{m+1}.p_{m-1,N-m+1} \\
&+ \frac{1}{m+1}.p_{m,N-m}
\end{aligned}$$

Therefore,

$$\begin{aligned}
p_{m,N-m+1} &= \frac{1}{m+1}\left(\frac{N}{2}+1+\sum_{k=0}^{m-N/2-1} p_{m-k,N-m+k}\right) \\
&\geq \frac{1}{m+1}\left(\frac{N}{2}+1+\sum_{k=0}^{m-N/2-1}\frac{1}{2}\right) \text{ from our inductive assumption} \\
&= \frac{1}{m+1}\left(\frac{N}{2}+1+\left(m-\frac{N}{2}\right)\frac{1}{2}\right) \\
&= \frac{N/2+2+m}{m+1}.\frac{1}{2}
\end{aligned}$$

Thus,

$$\begin{aligned} & p_{m,N-m+1} \geq \frac{1}{2} \\ \iff\quad & \frac{N/2+2+m}{m+1} \geq 1 \\ \iff\quad & N/2 \geq -1 \\ \iff\quad & N \geq -2 \end{aligned}$$

Hence, $p_{m,N-m+1} \geq \frac{1}{2}$.

Then, for the initial state,

$$\begin{aligned} p_{N,1} &= \frac{N}{N+1}.p_{N-1,2} \\ &\geq \frac{N}{N+1}.\frac{1}{N}.\left(\frac{N}{2}+1+\left(N-1-\frac{N}{2}\right)\frac{1}{2}\right) \\ &= \frac{1}{2}\frac{3N+2}{2N+2} \end{aligned}$$

Hence, $p_{N,1} \geq \frac{1}{2}$.

Therefore, by induction,

$$\forall N \geq 3, \forall r = (s,i) \text{ s.t. } s+i = N\ \mathcal{S}_N, r \models \mathcal{P}_{\geq 1/2}[\Diamond(i > s)].$$

Since, for $s = N-1, i = 1,\ s+i = N$,

$$\forall N \geq 3,\ \mathcal{S}_N \models \mathcal{P}_{\geq 1/2}[\Diamond(i > s)]$$

□

7.6 Discussion

A 'Continuum' of Degenerative Systems The distinction between degenerative and semi-degenerative families of models is not absolute and indeed depends on the property considered. For example, if we were to verify the property *"with probability 1 eventually the number of infectives equals zero"* then we could consider the SIR protocol as probabilistically degenerative since every model converges to the base system model (any system model for which the number of infectives equals zero) with probability 1. However, other system and property combinations may not conform to any of the degenerative definitions outlined in this thesis.

Convergent Properties In Section 6.3 we discussed the work of Duflot et al. [27, 28] and its relation to our work on degenerative systems. In the case of the SIR protocol population models, it would appear that this work is again applicable. In particular, we could define a measure on the states of the model based on the number of infectives and then employ the usual ordering on the natural numbers. It should be apparent that the measure is non-increasing and therefore, we can establish certain qualitative properties of the protocol. Again, however, we cannot consider quantitative properties such as *Property P2* described above.

Infinite-state Models of Probabilistic Systems It should be obvious that, rather than considering the models derived from our PRISM specification of the population version of the SIR protocol as a family, we could instead define them in terms of a single infinite-state model. In [43], the authors describe a sound but incomplete algorithm for determining the maximum probability of a reachability property holding in a infinite state MDPs. The method relies on establishing symbolic transition *types*. Since each model in the family of SIR protocol models has a transition distribution over states that does not correspond to a transition distribution in the smaller models, it is unclear whether this method would be applicable to our example.

Families of MDPs Probabilistic distributed systems with concurrently executing processes are normally modelled using a Markov Decision Process and therefore, ideally, our technique should be applicable to MDP models. In this chapter, on the other hand, we employ DTMC models. However, when considering probabilistic reachability (i.e. the probability of reaching a set of states in a MDP from some state) as we do in this chapter, it is only necessary to analyse *simple* adversaries [13]. For any finite path of a MDP, a simple adversary chooses a distribution based only on the last state of the path regardless of the states previously visited. Hence, the Markov chain associated with a simple adversary can be considered as a finite state DTMC. We can therefore apply the results obtained in this chapter to the DTMCs derived from simple adversaries. Thus, it should be possible to extend our results to families of MDPs.

Summary In this chapter we considered an approach for families of $DTMC$ models that 'converge' to a base system model with probability less than 1 i.e. for which with some non-zero probability it is possible to reach a 'terminal' state from which it is not possible to reach a state of the base system model. We formally define these models by considering a sub-chain relation that, intuitively, relates two models if one is 'embedded' in the other. Any family of models for which, for all N, the model of a system of size N is a sub-isomorphism of the model of a system of size $N+1$ are described as semi-degenerative. We illustrated that induction can be employed to prove properties of semi-degenerative systems by considering a simple example of a gossip protocol, which we specified in PRISM. We defined two types of specifications, one at the individual level and one at the population level. We verified certain properties of these for fixed sizes of system using PRISM and then tackled the parameterised model checking problem for the population specification using inductive reasoning.

Chapter 8

Open Problems

Outline In this chapter we highlight the limitations of our techniques by providing examples of systems for which our methods are not currently applicable and then suggest ways in which we can overcome these limitations.

8.1 A Replicated Databases Gossip Protocol

In Chapter 7 we described a class of protocols commonly referred to as gossip protocols. Here we consider one such protocol due to Demers et al. [25]. Demers et al. describe a gossip protocol for maintaining updates within a replicated distributed database environment that is based on a variation of the SIR model. A network site must maintain a list of 'infective' updates (updates to the database which it has made and wishes to spread to other sites). Any site with a non-empty list of updates chooses another site uniformly at random and transmits the list of updates. If that site is susceptible with respect to any of the updates, i.e. it has not yet received the update, it will make that update and add it to its infective list. If, however, the site has already received some update on the list, then the infecting site will, with some fixed probability, remove the item from its list.

We have developed a PRISM 'population' specification to model the rumour mongering replicated databases protocol (RDP) of Demers et al. The specification is very similar to that given for the SIR protocol in Appendix H except that it is for a MDP and the probabilities associated with the updates are different.

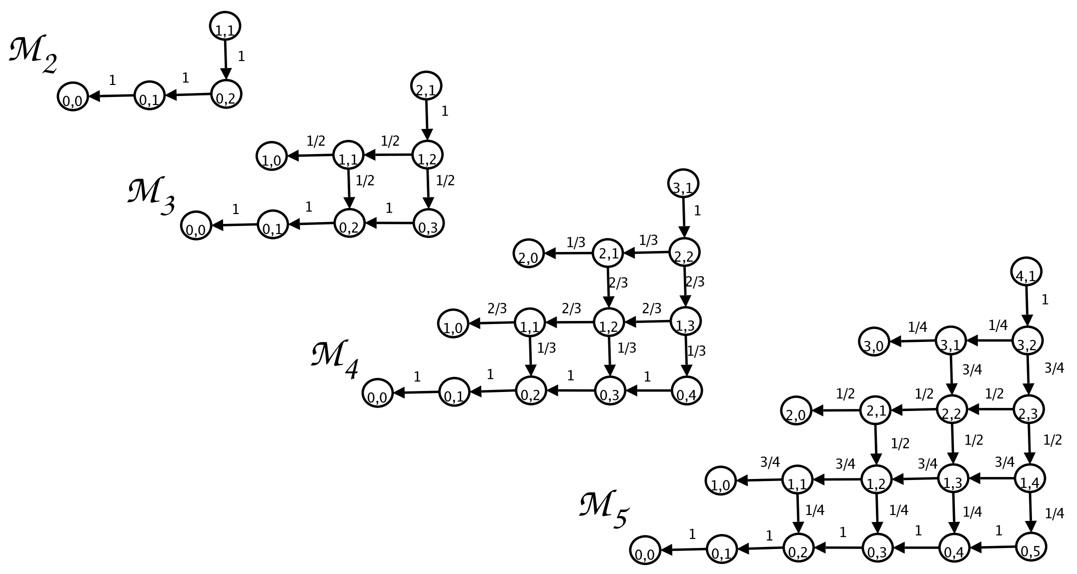


Figure 8.1: DTMCs for the replicated databases gossip protocol for 2,3,4 and 5 sites

Figure 8.1 shows the MDP models that would be generated from our PRISM population specification, for two, three, four and five sites and $k = 1$. Each state is labelled with two values. The first of these denotes the number of susceptible sites (that haven't yet received the update), and the second the number of infective sites (that have received the update and are still passing it on). Note that there is only one non-deterministic choice from each state and therefore we omit the dashed transition lines from the diagram. We emphasise at this point that the probabilities associated with the transitions in the model are dependent on N, the number of sites.

Consider the parameterised model checking problem for the RDP. We observe that, once a site becomes removed, it no longer actively participates in the protocol. It would appear, therefore, that the RDP is degenerative. However, on closer inspection, we realise that, once a site becomes removed, it is still possible for other infective sites to send messages to that site, whereby the infective site may then itself become removed. Therefore, a removed site will maintain a passive role in the protocol.

The degenerative behaviour of the protocol can be seen *structurally* in the underlying Markov chain (see Figure 8.1). It should be apparent that, ignoring the probabilities, the smaller models are, structurally speaking, almost subgraphs of the larger ones (with the exception of an additional transition from the state $(N-1, 1)$ to the state $(N-1, 0)$ in the larger model). However, unlike the SIR protocol, the transition probabilities change as N

changes. This is because the probabilities are dependent on N, the number of sites, which is in turn related to the fact that a removed site still passively participates in the protocol. In particular, notice that the 'downward' transition probabilities in the corresponding subgraph decrease as N increases, while the 'horizontal' transition probabilities increase with N. This means that the model $\mathcal{M}_N$ is not embedded in $\mathcal{M}_{N+1}$ and therefore the family of models associated with the population specification of the RDP cannot be semi-degenerative. This also makes it difficult to devise an invariant for the protocol in the manner of Chapter 4.

Verification of qualitative properties is independent of the actual probabilities in a model and therefore it may be appropriate to consider properties such as these, but note that the terminating behaviour of the protocol precludes proving properties which 'converge' to the base system since these inherently will hold with probability between 0 and 1. Nonetheless it may be possible to prove a property such as *"with probability 1 eventually the number of infectives is zero"* since this holds in every terminal state of the model.

This example illustrates well two of the limitations of the techniques described for degenerative systems in Chapters 5, 6 and 7. Namely, we require the behaviour of any degenerative system once it has 'degenerated' to *exactly* match that of a 'smaller' system and the behaviour of the system to be independent of the size of the system. In the case of the RDP, the transition probabilities are dependent on the size of the system and therefore the behaviour of a 'degenerated' system is different from that of any 'smaller' system.

8.2 A Weak Shared Coin Protocol

In Section 1.4 we discussed the weak shared coin protocol of Aspnes and Herlihy [6]. This protocol is used to return the result of a set of coin flips by a set of concurrently executing processes such that there is some bound on the probability of every process flipping heads (or tails). The protocol operates by employing a shared counter that each process can read from and write to. The size of the counter is determined by the number of processes, N and an independent parameter $K > 1$. Each process will flip a coin, choosing heads or tails with equal probability. If a process chooses heads then it will increment the counter and if it chooses tails it will decrement the counter. It will then read the counter and if the value lies between some upper and lower threshold values (determined by N and K)

then it will flip the coin again. If, on the other hand, the counter value is below the lower threshold, the process will choose tails and terminate, and similarly if the counter value is above the upper threshold it will choose heads and terminate.

Once a process has terminated, the protocol proceeds independently of that process. The protocol therefore appears to exhibit degenerative behaviour. One of the difficulties in formally establishing that the protocol is degenerative is due to the size of the counter, which is dependent on N. However, if only the local state of each of the processes is considered (as was the case for the Itai Rodeh protocol in Chapter 6), it may still be possible to prove that the protocol is degenerative. Note that the processes degenerate probabilistically rather than deterministically and therefore we would need to establish that the protocol was probabilistically (rather deterministically) degenerative. As an alternative, it would be interesting to investigate whether the techniques described in Chapter 7 for semi-degenerative systems could be extended to be applicable in this instance. This is further work.

8.3 Determining the Size of a Ring

In Chapter 6 we considered the Itai Rodeh protocol for determining a leader in an anonymous ring. To ensure correct operation of the protocol it is necessary for each process in the ring to know the size of the ring it is operating in. It has been shown that a leader election protocol that terminates with probability 1 exists only if the size of the ring is known to be within some bounds.

In some situations, this may not be known and must be determined. A randomised algorithm has been devised that determines the correct size of the ring with probability 1 if it is known that it lies in the bound $-2N, \ldots, 2N$ [35]. We consider an alternative algorithm that does not require any information about the size of the ring but that gives a guarantee of correctness with probability arbitrarily close to 1 [35]. Each process participating in the protocol chooses an id value of 0 or 1 with equal probability and then transmits this value along with the a counter which contains the highest counter value it has seen so far. If a process receives a message with id value equal to its own id then it updates its counter depending on the counter value received and then transmits a new message. Each process does this r times, where r is a constant parameter of the protocol.

We have defined PRISM specifications for the system described above based on the algorithm given in [35]. Note that channels are assumed to be FIFO and of length two (this is shown to be sufficient in [35]).

For this algorithm, for any size of ring N, in [35] it is proven that there is a lower bound of $2^{-Nr/2}$ for the probability of error in determining the ring size correctly (it is also shown that the actual value is $2^{-(N-1)r}$, where N is prime).

Parameterised verification of this protocol using the techniques described in the previous chapters is difficult. The complexity of the protocol inhibits the derivation of an invariant in the manner of Chapter 4 and the protocol does not exhibit degenerative behaviour in the manner prescribed in Chapters 5, 6 or 7. It would seem clear therefore that there are systems for which it is not straightforward to apply any of our methods.

Summary In this chapter we have considered some open problems that remain to be tackled. In particular, we considered three examples of systems that are not currently accomodated by any of our techniques: a replicated databases gossip protocol, a probabilistic protocol to determine the size of a ring and the weak shared coin protocol of Aspnes and Herlihy. Each of these systems displayed properties that deemed our inductive and abstraction techniques inapplicable. In some of these cases we provided suggestions for extending our techniques such that they could be applied. In others it would appear that new techniques are necessary.

Chapter 9

Conclusions

Outline This chapter concludes the thesis, providing an overview of the work described in the previous chapters.

9.1 Introduction

As discussed in Chapters 1 and 2, model checking is a useful verification technique for the formal analysis of distributed systems. Model checking tools verify that properties capturing the correct behaviour of some system are satisfied by a model of the system. Properties are usually specified using a temporal logic (Section 2.3) and models as Kripke structures (Section 2.2). Verification is generally done by searching the state-space of the model. However, the state-space of the model grows rapidly with the number of components being modelled (the *state-space explosion problem*) and so model checking tools must employ (a variety of) techniques to combat this problem.

In this thesis we have considered the verification of *randomised* distributed systems. *Probabilistic* model checking tools, such as PRISM (see Section 3.4), provide the means to perform *quantitative* as well as qualitative analysis of systems. PRISM enables the verification of properties, specified in a probabilistic temporal logic, against probabilistic models of a system, specified using PRISM. In particular, PRISM provides support for defining MDP models (described in Section 3.2.2) which display non-deterministic as well as probabilistic behaviours and are therefore an appropriate model for randomised distributed systems. These are verified with respect to properties described in the PCTL temporal logic (see Section 3.3).

One of the limitations of model checking of distributed systems is that it is only possible to consider systems that have a fixed number of components. The problem of verifying a distributed system for an arbitrary number of components, the parameterised model checking problem, was introduced in Chapter 1 along with a summary of some of the work that has been undertaken on tackling the problem. A considerable body of work exists on tackling the PMCP for non-probabilistic systems but the problem has not been widely considered for probabilistic systems. The PMCP is undecidable, implying that, in order to solve it, it is necessary to either consider methods that are sound but incomplete or consider classes of distributed systems for which the problem is decidable.

9.2 Network Invariants and Abstraction

In Chapter 4 we tackled the PMCP for a token ring protocol by constructing a *network invariant* using an approach based on *data abstraction*, employed previously for a number of non-probabilistic systems. We specified models of the protocol, and the invariant, using Promela and established that any properties that hold for the invariant are true for any model of the protocol for an arbitrary sized ring. Using model checking we proved that certain properties were satisfied by the invariant. Furthermore, we constructed an invariant for probabilistic models of the token ring protocol specified using PRISM. By adapting the proof for the non-probabilistic case, we established that certain properties that were true of the invariant were also true of every model of the protocol. Since a network invariant does not always exist [62], this approach is sound but incomplete.

We constructed our proof for the token ring protocol manually. It would be interesting to establish whether we could automate the proof, for example by extending an automatic invariant technique of [39] to the probabilistic domain. Furthermore, it would be convenient to define an abstract specification from any given concrete specification, using a set of generic restrictions to guide the construction. If suitable restrictions were used then the abstract specification would be guaranteed to be a valid abstraction of the concrete specification. Hence, an extension of this work is to analyse the use of abstraction, in order to determine whether such restrictions exist and if so, what they are.

9.3 Degenerative Systems

In Chapters 5 and 6 we solved the PMCP by considering a restriction on the types of systems that we analyse. Specifically, we provided a technique for reasoning about a class of parameterised probabilistic systems described as degenerative. We distinguished between deterministically and probabilistically degenerative families of MDP models for a system. Under a particular scheduler a deterministically degenerative system model will degenerate to some system model with probability 1. For a probabilistically degenerative system on the other hand, under some scheduler the system model will degenerate to some *set* of system models with probability 1.

We presented inductive proof schemas for reasoning about these two classes of systems, generalising an inductive proof that was introduced for a non-probabilistic parameterised distributed system [49]. The proof schema provides a method for establishing that any property that holds of a model of a system with some base topology will hold for a model of any system size and configuration. The technique relies on showing that any behaviour in a model of the system of a given size is (observationally) 'equivalent' to a behaviour in a model of a smaller system.

We employed two case studies to demonstrate our method. In Chapter 5 we considered a family of models of the IEEE 1394 Firewire tree identify protocol [33] and provided an inductive proof over the system topology (trees) to show that a certain class of probabilistic temporal logic properties that are satisfied by a model of a system with a star topology are satisfied by a model of a system with any acyclic topology. In Chapter 6 we considered the Itai Rodeh leader election protocol for rings and proved a similar result, over the set of rings.

In Chapter 7 we considered an extension to degenerative systems, by defining a semi-degenerative family of models of a system. Models of this type degenerate but may also terminate before reaching a model of a smaller system. We gave an example of a simple gossip protocol, showing that the family of models is semi-degenerative. We then exploited the semi-degenerative property and employed induction to prove properties of the protocol for an arbitrary number of network sites.

9.4 Open Problems

Finally, in Chapter 8 we described three examples of parameterised randomised distributed systems for which none of the approaches described above were applicable. Future work is to extend our techniques to deal with these examples.

Summary We have provided a summary of the work described in the thesis, outlining the approaches we have considered for tackling the parameterised model checking problem.

Bibliography

[1] PRISM home page. `http://www.prismmodelchecker.org/`.

[2] R. Alur and T. A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1), 1999.

[3] D. Angluin. Local and global properties in networks of processors (extended abstract). In *Proceedings of the Twelfth annual ACM symposium on Theory of Computing (STOC'80)*, pages 82–93. ACM Press, 1980.

[4] K. R. Apt and D. C. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22:307–309, 1986.

[5] T. Arons, A. Pnueli, and L. D. Zuck. Parameterized verification by probabilistic abstraction. In *Proceedings of the Sixth International Conference on Foundations of Software Science and Computational Structures (FoSSaCS'03).*, volume 2620 of *Lecture Notes in Computer Science*, pages 87–102. Springer, 2003.

[6] J. Aspnes and M. Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 11(3):441–461, 1990.

[7] C. Baier, F. Ciesinski, and M. Groesser. Quantitative analysis of distributed randomized protocols. In *Proceedings of the Tenth International Workshop on Formal Methods for Industrial Critical Systems (FMICS'05)*, pages 2–7. ACM Press, 2005.

[8] C. Baier, M. Groser, and F. Ciesinski. Partial order reduction for probabilistic systems. In *Proceedings of the First International Conference on The Quantitative Evaluation of Systems (QEST'04)*, pages 230–239. IEEE Computer Society, 2004.

[9] C. Baier, H. Hermanns, J.-P. Katoen, and V. Wolf. Comparative branching-time semantics for Markov chains. *Information and Computation*, 200:149–214, 2005.

[10] C. Baier and M. Kwaitkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998.

[11] N.T.J. Bailey. *The Mathematical Theory of Infectious Diseases and its Applications.* Griffin, London, second edition, 1975.

[12] I. Balaban, A. Pnueli, and L. D. Zuck. Invisible safety of distributed protocols. In *Proceeding of the 33rd International Colloquium on Automata, Languages and Programming (ICALP'06), Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 528–539. Springer, 2006.

[13] A. Bianco and L. de Alfaro. Model checking of probabalistic and nondeterministic systems. In *Proceedings of the Fifteenth Conference on Foundations of Software Technology and Theoretical Computer Science (FST TCS'95)*, pages 499–513. Springer-Verlag, 1995.

[14] P. Billingsley. *Probability & Measure.* Springer-Verlag, New York, first edition, 1979.

[15] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications.* Macmillan Press, London and Basingstoke, 1978.

[16] M. Calder and A. Miller. Using SPIN to analyse the tree identification phase of the IEEE 1394 high performance serial bus (FireWire) protocol. In Maharaj and Shankland [46], pages 247–266.

[17] M. Calder and A. Miller. Detecting feature interactions: how many components do we need? In *Objects, agents and features*, Lecture Notes in Computing Science, pages 45–66. Springer-Verlag, 2004.

[18] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking.* The MIT Press, Cambridge, Masachusetts, 1999.

[19] E.M. Clarke, O. Grumberg, and S. Jha. Verifying parameterized networks using abstraction and regular languages. In *Proceedings of the Sixth International Conference on Concurrency Theory (CONCUR '95)*, volume 962 of *Lecture Notes in Computer Science*, pages 395–407. Springer-Verlag, August 1995.

[20] E.M. Clarke, O. Grumberg, and D Long. Model checking and abstraction. In *Conference Record of the Nineteenth Annual ACM Symposium on Principles of Programming Languages (POPL '92)*, pages 343–354. ACM Press, January 1992.

[21] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252. ACM Press, 1977.

[22] S. J. Creese and A. W. Roscoe. Verifying an infinite family of inductions simultaneously using data independence and FDR. In *Proceedings of Formal Techniques for Networked and Distributed Systems (FORTE'99)*, volume 156 of *IFIP Conference Proceedings*, pages 437–452. Kluwer, 1999.

[23] S.J. Creese and A.W. Roscoe. Data independent induction over structured networks. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'00)*, volume II. CSREA Press, 2000.

[24] P. D'Argenio, B. Jeannet, H.E. Jensen, and K.G. Larsen. Reduction and refinement strategies for probabilistic analysis. In *Process Algebra and Probabilistic Methods - Performance Modelling and Verification (PAPM-PROBMIV'02)*, volume 2399 of *Lecture Notes in Computer Sciences*, 2002.

[25] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth Annual ACM Symposium on Principles of distributed computing (PODC'87)*, pages 1–12. ACM Press, August 1987.

[26] A. Donaldson and A. Miller. Symmetry reduction for probabilistic model checking using generic representatives. In *Proceedings of the fourth International Symposium on Automated Technology for Verification and Analysis (ATVA'06)*, volume 4218 of *Lecture Notes in Computer Science*, pages 9–23. Springer, 2006.

[27] M. Duflot, L. Fribourg, and C. Picaronny. Randomized finite-state distributed algorithms as Markov chains. In *Distributed Algorithms*, volume 2180 of *Lecture Notes in Computer Science*, pages 240–254, 2001.

[28] M. Duflot, L. Fribourg, and C. Picaronny. Randomized dining philosophers without fairness assumption. *Distributed Computing*, 17(1):65–76, 2004.

[29] E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In *Conference Record of the 22nd Annual ACM Symposium on Principles of Programming Languages (POPL '95)*, pages 85–94. ACM Press, 1995.

[30] W. Fokkink and J. Pang. Variations on Itai-Rodeh leader election for anonymous rings and their analysis in PRISM. *Journal of Universal Computer Science*, 12(8):981–1006, 2006.

[31] D. Graham, M. Calder, and A. Miller. An inductive technique for parameterised model checking of degenerative distributed randomised protocols. In *Proceedings of the Seventh International Workshop on Automated Verification of Critical Systems (AVoCS'07)*, Electronic Notes in Theoretical Computer Science. Elsevier ScienceDirect, 2007. To appear.

[32] G. J. Holzmann. *The SPIN model checker: primer and reference manual.* Addison Wesley, Boston, 2003.

[33] IEEE 1394-1995. *IEEE Standard for a High Performance Serial Bus Std 1394-1995.* Institute of Electrical and Electronic Engineers, August 1995.

[34] A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *Proceedings of the ninth annual ACM symposium on Principles of distributed computing (PODC'90)*, pages 119–131. ACM Press, 1990.

[35] A. Itai and M. Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1):60–87, 1990.

[36] B. Jeannet, P. R. D'Argenio, and K. G. Larsen. RAPTURE: A tool for verifying Markov Decision Processes. In *Tools Day, International Conference on Concurrency Theory, (CONCUR'02)*, 2002. Technical Report, Faculty of Informatics at Masaryk University Brno.

[37] J. G. Kemeny, J. L. Snell, and A. W. Knapp. *Denumerable Markov Chains*, volume 40 of *Graduate Texts in Mathematics.* Springer-Verlag, New York, second edition, 1976.

[38] Y. Kesten and A. Pnueli. Control and data abstraction: The cornerstones of practical formal verification. *International Journal on Software Tools for Technology Transfer (STTT)*, 2(4):328–342, 2000.

[39] Y. Kesten, A. Pnueli, E. Shahar, and L. D. Zuck. Network invariants in action. In *Proceedings of the Thirteenth International Conference on Concurrency Theory (CONCUR '02)*, pages 101–115. Springer-Verlag, 2002.

[40] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(2):128–142, 2004.

[41] M. Kwiatkowska, G. Norman, and D. Parker. Game-based abstraction for Markov Decision Processes. In *Proceedings of the Third international conference on the Quantitative Evaluation of Systems (QEST'06)*, pages 157–166. IEEE Computer Society, 2006.

[42] M. Kwiatkowska, G. Norman, and D. Parker. Symmetry reduction for probabilistic model checking. In *Proceedings of the Eighteenth International Conference on Computer Aided Verification (CAV'06)*, volume 4114 of *Lecture Notes in Computer Science*, pages 234–248. Springer-Verlag, 2006.

[43] M. Kwiatkowska, G. Norman, and J. Sproston. Symbolic computation of maximal probabilistic reachability. In *Proceedings of the Thirteenth International Conference on Concurrency Theory (CONCUR'01)*, volume 2154 of *Lecture Notes in Computer Science*, pages 169–183. Springer, 2001.

[44] L. Lamport. What good is temporal logic? *Information Processing*, 83:657–668, 1983.

[45] R. Lazic and B. Roscoe. Data independence with generalised predicate symbols. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, pages 319–326. CSREA Press, 1999.

[46] J. Maharaj, S. Romijn and C. Shankland, editors. *Formal Aspects of Computing: Special issue on IEEE 1394 Tree Identification Protocol*, volume 14(3). Springer-Verlag, 2003.

[47] K.L. McMillan. Getting started with SMV. `http://www.cis.ksu.edu/santos/smv-doc/tutorial/tutorial.html`.

[48] S. Merz. Model checking: A tutorial overview. In *Proceedings of the Fourth Summer School on Modeling and Verification of Parallel Processes (MOVEP'00)*, volume 2067 of *Lecture Notes in Computer Science*, pages 3–38. Springer, 2000.

[49] A. Miller and M. Calder. Two verification results for networks of arbitrary size. Technical Report TR2006-220, University of Glasgow,Department of Computing Science, 2006.

[50] G. Norman. Analyzing randomized distributed algorithms. In *Validation of Stochastic Systems: A Guide to Current Research*, volume 2925 of *Lecture Notes in Computer Science (Tutorial Volume)*, pages 384–418. Springer, 2004.

[51] S. Owre, N. Shankar, and J. Rushby. PVS: A prototype verification system. In *Automated Deduction - Proceedings of the Eleventh International Conference on Automated Deduction (CADE'92)*, volume 607 of *Lecture Notes in Computer Science*, pages 748–752. Springer-Verlag, June 1992.

[52] A. Pnueli and E. Shahar. Liveness and acceleration in parameterized verification. In *Proceedings of the Twelfth International Conference on Computer Aided Verification (CAV'00)*, pages 328–343. Springer-Verlag, 2000.

[53] A. Pnueli, J. Xu, and L. D. Zuck. Liveness with (0, 1, ∞)-counter abstraction. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV'02)*, pages 107–122. Springer-Verlag, 2002.

[54] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley and Sons, New York, first edition, 1994.

[55] J.J.M.M Rutten, M. Kwiatkowska, G. Norman, and D. Parker. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems*, volume 23 of *CRM Monograph Series.* American Mathematical Society, Providence, Rhode Island, 2004.

[56] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, MIT, Dept. of Electrical Engineering and Computer Science, 1995.

[57] R. Segala. Verification of randomized distributed algorithms. *Lectures on formal methods and performance analysis: first EEF/Euro summer school on trends in computer science*, 2090:232–260, 2001.

[58] R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[59] J. Song and O.W.W. Yang. Rotation counter protocol - a token-ring protocol for integrated services. In *Proceedings of the Eighteenth Conference on Local Computer Networks (LCN'93)*, pages 474–481, 1993.

[60] M. Stoelinga. Fun with firewire: A comparative study of formal verification methods applied to the IEEE 1394 root contention protocol. *Formal Aspects of Computing*, 14(3):328–337, 2003.

[61] Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FoCS'85)*, pages 327–338. IEEE, 1985.

[62] 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 (AVMFSS'90)*, volume 407 of *Lecture Notes in Computer Science*, pages 68–80. Springer-Verlag, 1990.

Appendix A

Promela Specification for Counter Token Ring Protocol

```
/************************************************************
 *
 * COUNTER TOKEN RING PROTOCOL (CONCRETE VERSION)
 *
 * Several processes pass a token counter between them.
 * When work is true, token counter is decremented
 * When work is reset the protocol recommences.
 *
 ************************************************************/

#define N 4 //no of processes
#define c 5 //initial value of counter

//define channels for token to be passed on (ch[1] = process 0
//to process 1, ch[3] = process 2 to process 3 etc..)
chan ch[N] = [1] of {int};

bool work=false; // whether process should work
bool done=false; // once work is false, can't be reset until !done
```

```
     // and once work is true, can’t be set false until done
bool end=false; // when the token=0 this is set to true
int finished=0; // add 1 to finished when token decremented
                //(only used for verifying properties)



/*****************************************************************
 * Arbiter process
 */
proctype arbiter()
{
change:
    if
    :: atomic{(done && work)-> work=false; goto change;}  // 7.
    :: atomic{(!done && !work)-> work=true; goto change;} // 8.
    fi;
}//arbiter


/****************************************************************/

proctype process(chan in,out;int id)
{
int token=0; // stores counter value
bool possess=false;// set possess to true when token is received,
             // (only used for verifying properties)
bool working=false;// set working to true when process does work
   // set back to true once done to pass token (for verification)

idle:
    do
      ::atomic{full(in)->in?token; possess=true;} // 1. accept token
```

```
rcvd:   if
        ::atomic{!work && empty(out) -> // 2. pass token
            out!token;
            possess=false;}
        ::atomic{work && token>1 && empty(out)->//3.decrement token
            working=true;
            token--;
            done=true;
            fi;}
pass:       atomic{!work && empty(out) -> // 4. pass token
                out!token;
                possess=false;
                done=false;
                working=false;}
        ::atomic{work && token==1 ->  // 5. finished
            working=true;
            token--;
       finished++; end = true; goto finish;
        fi;
    od;
finish:
    goto finish;  // 6. finished so loop
}//process

/***************************************************************/

init{
  atomic{                 // 0. initialise protocol
        run process(ch[0],ch[1],0);
        run process(ch[1],ch[2],1);
        run process(ch[2],ch[3],2);
        run process(ch[3],ch[0],3);
        run arbiter();
        ch[0]!c; // always start with process 0
   }
}//init

/************************************************************/
```

Appendix B

Promela Invariant Specification for Counter Token Ring Protocol

```
/***********************************************************
 *
 * COUNTER TOKEN RING PROTOCOL (INVARIANT VERSION)
 *
 * Several processes pass a counter token between them.
 * When the arbiter process decides, the counter is decremented
 * and the protocol then recommences.
 * Protocol finishes when process has token that
 * equals zero.
 *
 *
 ***********************************************************/

#define m 3 //no of concrete processs+1
#define c 5 //initial value of counter

// channels for token to be passed on (ch[0] = process N to process 0,
// ch[1] = process 0 to process 1, ch[2] = process 1 to process 2 etc.)
```

```
chan ch[m] = [1] of {int};

bool work=false;  // whether process should work
bool done=false;  // once work is false, can’t be reset until !done
         // and once work is true it can’t be set false until done
bool end=false;  // when the token=0 this is set to true
int finished=0;  // add 1 to finished when token decremented
                 //(only used for verifying properties)

/*****************************************************************
 * Arbiter process
 */
proctype arbiter()
{
change:
    if
    :: atomic{(done && work)-> work=false; goto change;}  // 7.
    :: atomic{(!done && !work)-> work=true; goto change;} // 8.
    fi;
}//arbiter


/*****************************************************************/

proctype process(chan in,out;int id)
{
int token=0; // stores counter value
bool possess=false;// set possess to true when token is received,
             // (only used for verifying properties)
bool working=false;// set working to true when process does work
  // set back to true once done to pass token (for verification)

idle:
    do
      ::atomic{full(in)->in?token; possess=true;} // 1. accept token
rcvd:   if
        ::atomic{!work && empty(out) -> // 2. pass token
            out!token;
            possess=false;}
        ::atomic{work && token>1 && empty(out)->//3.decrement token
            working=true;
            token--;
            done=true;
            fi;}
pass:       atomic{!work && empty(out) -> // 4. pass token
                out!token;
                possess=false;
                done=false;
                working=false;}
        ::atomic{work && token==1 ->  // 5. finish
            working=true;
            token--;
       finished++; end = true; goto finish;
        fi;
    od;
finish:
    goto finish;  // 6. finished so loop
}//process


/*************************************************************/

proctype abstract_process(chan in,out;int id)
{
int token=0; // abstract token counter
```

```
idle:
    do
      ::atomic{full(in)->in?token;} // 1. accept token
rcvd:   if
        ::atomic{!work && empty(out) ->// 2.(a) pass token
            out!token;}
        ::atomic{!work && empty(in) -> // 2.(b) keep token
            in!token;}
        ::atomic{work && token>1 -> // 3. decrement counter
            working=true;
            token--;
            done=true;}
            if
pass:       ::atomic{!work && empty(out) ->  // 4.(a)
                out!token;          //pass token to neighbour
                done=false;}
            ::atomic{!work && empty(in) ->  // 4. (b)
                in!token;           //pass token back to self
                done=false;}
            fi;
        ::atomic{work && token==1 -> // 5. finish
            token--;
       finished++; end = true; goto finish;
        fi;
    od;
finish:
    goto finish;  // 6. finished so loop
}//abstract_process


/****************************************************************/


init{
  atomic{                      // 0. initialise protocol
        run process(ch[0],ch[1],0);
        run process(ch[,ch[2],1);
        run abstract_process(ch[2],ch[0],2);
        run arbiter();

        ch[0]!c;
    }
}//init

/**************************************************************/
```

Appendix C

PRISM Specification for Counter Token Ring Protocol

```
nondeterministic

 const int c=5; // initial value of counter assumed to be at least 1
 const int N=4; // number of processes

 const int idle=0; const int rcvd=1;
 const int pass=2; const int finish=3;

 const int empty=0;

 global ch0:[0..c] init c; global ch1:[0..c] init empty;
 global ch2:[0..c] init empty; global ch3:[0..c] init empty;

 global end : bool init false;
 global finished : [0..N] init 0;


module process0
```

```
 possess0 : bool init false;
 working0 : bool init false;
 loc0 : [idle..finish] init idle;
 token0 : [0..c] init 0;

 // receive token
 [] (loc0=idle & ch0!=empty) -> \\ 1.
   1: (possess0'=true) & (ch0'=empty) & (loc0'=rcvd) & (token0'=ch0);

 // randomly choose to work or just pass on token
 // when decrementing counter, if it reaches zero, protocol terminates
 // otherwise it is passed on
 [] (loc0=rcvd & ch1=empty & token0=1) -> \\ 2.
 0.5:(loc0'=finish)&(working0'=true)&(token0'=token0-1)
     &(finished'=finished+1)&(end'=true)
 +0.5: (loc0'=idle) & (possess0'=false) & (ch1'=token0) & (token0'=0);
 [] (loc0=rcvd & ch1=empty & token0>1) -> \\ 3.
  0.5: (working0'=1) & (token0'=token0-1) & (loc0'=pass)
 +0.5: (loc0'=idle) & (possess0'=false) & (ch1'=token0) & (token0'=0);

 // pass on token having decremented token counter
 [] (loc0=pass & ch1=empty) -> \\ 4.
 1:(loc0'=idle)&(ch1'=token0)&(possess0'=false)
   &(working0'=false)&(token0'=0);

 // counter has reached zero so process can finish
 [] (loc0=finish) -> (loc0'=loc0); \\ 5.

endmodule



module process1=process0[loc0=loc1, token0=token1,
 possess0=possess1, working0=working1, ch0=ch1, ch1=ch2] endmodule
```

```
module process2=process0[loc0=loc2, token0=token2,
 possess0=possess2, working0=working2, ch0=ch2, ch1=ch3] endmodule
module process3=process0[loc0=loc3, token0=token3,
 possess0=possess3, working0=working3, ch0=ch3, ch1=ch0] endmodule
```

Appendix D

PRISM Invariant Specification for Counter Token Ring Protocol

```
nondeterministic

 const int c=5; // initial value of counter, assumed to be at least 1
 const int N=4; // number of processes

 const int idle=0; const int rcvd=1;
 const int pass=2; const int finish=3;

 const int empty=0;

 global ch0:[0..c] init c; global ch1:[0..c] init empty;
 global ch2:[0..c] init empty;

 global end : bool init false;
 global finished : [0..N] init 0;


module process0
```

```
 possess0 : bool init false;
 working0 : bool init false;
 loc0 : [idle..finish] init idle;
 token0 : [0..c] init 0;

 // receive token
 [] (loc0=idle & ch0!=empty) -> \\ 1.
   1: (possess0'=true) & (ch0'=empty) & (loc0'=rcvd) & (token0'=ch0);

 // randomly choose to work or just pass on token
 // when decrementing counter if it reaches zero protocol terminates
 // otherwise it is passed on
 [] (loc0=rcvd & ch1=empty & token0=1) -> \\ 2.
 0.5:(loc0'=finish)&(working0'=true)&(token0'=token0-1)
     &(finished'=finished+1)&(end'=true)
 +0.5: (loc0'=idle) & (possess0'=false) & (ch1'=token0) & (token0'=0);
 [] (loc0=rcvd & ch1=empty & token0>1) -> \\ 3.
  0.5: (working0'=true) & (token0'=token0-1) & (loc0'=pass)
 +0.5: (loc0'=idle) & (possess0'=false) & (ch1'=token0) & (token0'=0);

 // pass on token having decremented token counter
 [] (loc0=pass & ch1=empty) -> \\ 4.
 1:(loc0'=idle)&(ch1'=token0)&(possess0'=false)
   &(working0'=false)&(token0'=0);

 // counter has reached zero so process is finished
 [] (loc0=finish) -> (loc0'=loc0); \\ 5.

endmodule



module process1=process0[loc0=loc1, token0=token1,
 possess0=possess1, working0=working1, ch0=ch1, ch1=ch2] endmodule
```

```
module abstract_process2

 loc2 : [idle..finish] init idle;
 token2 : [0..c] init 0;

 // receive token
 [] (loc2=idle & ch2!=empty) -> \\ 1.
  1: (loc2'=rcvd) & (ch2'=empty) & (token2'=ch2);

 // randomly choose to work or just pass on token
 // when decrementing token, if it is zero protocol terminates
 [] (loc2=rcvd & ch1=empty & token2=1) -> \\ 2.1
0.5: (loc2'=finish)&(token2'=token2-1)
     &(finished'=finished+1)&(end'=true)
 +0.5: (loc2'=idle) & (ch1'=token2) & (token2'=0);
 [] (loc2=rcvd & ch1=empty & token2>1) -> \\ 3.1
  0.5: (token2'=token2-1) & (loc2'=pass)
 +0.5: (loc2'=idle) & (ch1'=token2) & (token2'=0);

 // abstract process can non-deterministically choose
 // to keep token by sending on abstract channel
 [] (loc2=rcvd & ch2=empty & token2=1) -> \\ 2.2
  0.5: (loc2'=finish)&(token2'=token2-1)
       &(finished'=finished+1)&(end'=true)
 +0.5: (loc2'=idle) & (ch2'=token2) & (token2'=0);
 [] (loc2=rcvd & ch2=empty & token2>1) -> \\ 3.2
  0.5: (token2'=token2-1) & (loc2'=pass)
 +0.5: (loc2'=idle) & (ch2'=token2) & (token2'=0);

 // pass on token having decremented token counter
 // (abstract process can choose to pass to itself)
 [] (loc2=pass & ch1=empty) -> \\ 4.1
  1: (loc2'=idle) & (ch1'=token2) & (token2'=0);
 [] (loc2=pass & ch2=empty) -> \\ 4.2
  1: (loc2'=idle) & (ch2'=token2) & (token2'=0);

 // counter has reached zero and process is finished
 [] (loc2=finish) -> (loc2'=finish); \\ 5.

endmodule
```

Appendix E

PRISM Specification for TIP

```
nondeterministic

const int start=0;
const int child_handshake=1;
const int parent_handshake=2;
const int handshakes_complete=3;
const int response=4;
const int contention=5;
const int winner=6;
const int loser=7;
const int become_child=8;
const int finish=9;

const int empty=0;
const int be_my_parent=1;
const int be_my_child=2;
const int ack=3;

global elected : [0..N] init 0;

const int N=3;
```

```
  const int v0=1;
  const int v1=2;
  const int v2=1;
  const int nodeid0=1;
  const int nodeid1=2;
  const int nodeid2=3;
  global toss0 : [0..2] init 0;
  global toss1 : [0..2] init 0;
  global toss2 : [0..2] init 0;
  global zeroone : [empty..ack] init empty;
  global onezero : [empty..ack] init empty;
  global onetwo : [empty..ack] init empty;
  global twoone : [empty..ack] init empty;

module node0

  position0 : [start..finish] init start;
  child0_0 : [0..N] init 0;
  child0_1 : [0..N] init 0;
  child0_2 : [0..N] init 0;
  adj0 : [0..v0] init v0;
  remaining_partner0 : [0..N] init 0;
  no_of_requests0 : [0..N] init 0;

  [] (position0=start & onezero=be_my_parent) ->
   (position0'=child_handshake) & (no_of_requests0'=1)
   & (onezero'=empty) & (child0_0'=nodeid1); // 1.
  [] (position0=start & onezero=empty) ->
   (position0'=child_handshake) & (no_of_requests0'=0)
   & (remaining_partner0'=nodeid1); // 4.

  [] (position0=child_handshake & adj0=v0
   & remaining_partner0=0 & zeroone=empty) ->
   (position0'=parent_handshake) & (zeroone'=be_my_child); // 2.
  [] (position0=child_handshake & adj0=v0 &
   remaining_partner0=nodeid1 & zeroone=empty) ->
   (position0'=parent_handshake) & (zeroone'=be_my_parent); // 5.

  [] (position0=child_handshake & adj0=1 & no_of_requests0=1
   & child0_0=nodeid1 & zeroone=empty) ->
   (position0'=parent_handshake) & (zeroone'=be_my_child); // 16.

  [] (position0=parent_handshake & adj0=v0 & remaining_partner0=0
   & onezero=ack) ->
   (position0'=finish) & (no_of_requests0'=0) & (elected'=nodeid0) &
   (adj0'=0)& (child0_0'=0)& (child0_1'=0)& (child0_2'=0) & (onezero'=empty); // 3.
  [] (position0=parent_handshake & adj0=v0 & remaining_partner0=nodeid1) ->
   (position0'=handshakes_complete); // 6.

  [] (position0=parent_handshake & adj0=1 & no_of_requests0=1
   & child0_0=nodeid1 & onezero=ack) ->
   (position0'=finish) & (onezero'=empty) & (elected'=nodeid0)
   & (no_of_requests0'=0) & (child0_0'=0) & (adj0'=0); // 17.

  [] (position0=handshakes_complete & no_of_requests0=adj0-1)
   -> (position0'=response); // 7.

  [] (position0=response & remaining_partner0=nodeid1
   & onezero=be_my_child) ->
   (position0'=become_child) & (remaining_partner0'=0)
   & (onezero'=empty); // 8.
  [] (position0=response & remaining_partner0=nodeid1
   & onezero=be_my_parent) ->
   (position0'=contention) & (onezero'=empty); // 10.
```

```
// contention (probabilistic choice always
// made by nodeid with smallest id)
  [] (nodeid0<nodeid1 & position0=contention
   & remaining_partner0=nodeid1 & toss0=0) ->
   0.25 : (position0'=winner) & (toss0'=1)
   + 0.25 : (position0'=loser) & (toss0'=2)
   + 0.5 : (toss0'=0); // 11.
  [] (nodeid0>nodeid1 & position0=contention
   & remaining_partner0=nodeid1 & toss1=1) ->
   (position0'=loser) & (toss1'=0); // 13.
  [] (nodeid0>nodeid1 & position0=contention
   & remaining_partner0=nodeid1 & toss1=2) ->
   (position0'=winner) & (toss1'=0); // 12.

// winner
  [] (position0=winner & remaining_partner0=nodeid1 & zeroone=empty) ->
   (zeroone'=be_my_parent) & (position0'=response); // 14.

// loser
  [] (position0=loser & remaining_partner0=nodeid1 & onezero=be_my_parent) ->
   (onezero'=empty) & (position0'=child_handshake) & (adj0'=1)
   & (remaining_partner0'=0) & (no_of_requests0'=1) & (child0_0'=nodeid1)
   & (child0_1'=0) & (child0_2'=0); // 15.

  [] (position0=become_child & zeroone=empty) ->
   (position0'=finish) & (zeroone'=ack) & (no_of_requests0'=0)
   & (remaining_partner0'=0) & (adj0'=0) & (child0_0'=0)
   & (child0_1'=0) & (child0_2'=0); // 9.

  [] (position0=finish& position1=finish& position2=finish) ->
   (position0'=finish);

endmodule
```

```
module node1

  position1 : [start..finish] init start;
  child1_0 : [0..N] init 0;
  child1_1 : [0..N] init 0;
  child1_2 : [0..N] init 0;
  adj1 : [0..v1] init v1;
  remaining_partner1 : [0..N] init 0;
  no_of_requests1 : [0..N] init 0;

  [] (position1=start & zeroone=be_my_parent & twoone=be_my_parent) ->
   (position1'=child_handshake) & (no_of_requests1'=2)
   & (zeroone'=empty) & (twoone'=empty) & (child1_0'=nodeid0)
   & (child1_1'=nodeid2); // 1.
  [] (position1=start & zeroone=empty & twoone=be_my_parent) ->
   (position1'=child_handshake) & (no_of_requests1'=1)
   & (remaining_partner1'=nodeid0) & (twoone'=empty)
   & (child1_0'=nodeid2); // 4.
  [] (position1=start & zeroone=be_my_parent & twoone=empty) ->
   (position1'=child_handshake) & (no_of_requests1'=1)
   & (remaining_partner1'=nodeid2) & (zeroone'=empty)
   & (child1_0'=nodeid0); // 4.

  [] (position1=child_handshake & adj1=v1 & remaining_partner1=0
   & onezero=empty & onetwo=empty) ->
   (position1'=parent_handshake) & (onezero'=be_my_child)
   & (onetwo'=be_my_child); // 2.
  [] (position1=child_handshake & adj1=v1 & remaining_partner1=nodeid0
   & onezero=empty & onetwo=empty) ->
   (position1'=parent_handshake) & (onezero'=be_my_parent)
   & (onetwo'=be_my_child); // 5.
```

```
[] (position1=child_handshake & adj1=v1 & remaining_partner1=nodeid2
 & onezero=empty & onetwo=empty) ->
 (position1'=parent_handshake) & (onezero'=be_my_child)
 & (onetwo'=be_my_parent); // 5.

[] (position1=child_handshake & adj1=1 & no_of_requests1=1
 & child1_0=nodeid0 & onezero=empty) ->
 (position1'=parent_handshake) & (onezero'=be_my_child); // 16.
[] (position1=child_handshake & adj1=1 & no_of_requests1=1
 & child1_0=nodeid2 & onetwo=empty) ->
 (position1'=parent_handshake) & (onetwo'=be_my_child); // 16.

[] (position1=parent_handshake & adj1=v1 & remaining_partner1=0
 & zeroone=ack & twoone=ack) ->
 (position1'=finish) & (no_of_requests1'=0) & (elected'=nodeid1)
 & (adj1'=0)& (child1_0'=0)& (child1_1'=0)& (child1_2'=0)
 & (zeroone'=empty) & (twoone'=empty); // 3.
[] (position1=parent_handshake & adj1=v1 & remaining_partner1=nodeid0
 & twoone=ack) ->
 (position1'=handshakes_complete) & (twoone'=empty); // 6.
[] (position1=parent_handshake & adj1=v1 & remaining_partner1=nodeid2
 & zeroone=ack) ->
 (position1'=handshakes_complete) & (zeroone'=empty); // 6.

[] (position1=parent_handshake & adj1=1 & no_of_requests1=1
 & child1_0=nodeid0 & zeroone=ack) ->
 (position1'=finish) & (zeroone'=empty) & (elected'=nodeid1)
 & (no_of_requests1'=0) & (child1_0'=0) & (adj1'=0); // 17.
[] (position1=parent_handshake & adj1=1 & no_of_requests1=1
 & child1_0=nodeid2 & twoone=ack) ->
 (position1'=finish) & (twoone'=empty) & (elected'=nodeid1)
 & (no_of_requests1'=0) & (child1_0'=0) & (adj1'=0); // 17.
```

```
  [] (position1=handshakes_complete & no_of_requests1=adj1-1) ->
   (position1'=response); // 7.

  [] (position1=response & remaining_partner1=nodeid0
   & zeroone=be_my_child) ->
   (position1'=become_child) & (remaining_partner1'=0)
   & (zeroone'=empty); // 8.
  [] (position1=response & remaining_partner1=nodeid0
   & zeroone=be_my_parent) ->
   (position1'=contention) & (zeroone'=empty); // 10.
  [] (position1=response & remaining_partner1=nodeid2
   & twoone=be_my_child) ->
   (position1'=become_child) & (remaining_partner1'=0)
   & (twoone'=empty); // 8.
  [] (position1=response & remaining_partner1=nodeid2
   & twoone=be_my_parent) ->
   (position1'=contention) & (twoone'=empty); // 10.

// contention (probabilistic choice always
// made by nodeid with smallest id)
  [] (nodeid1<nodeid0 & position1=contention
   & remaining_partner1=nodeid0 & toss1=0) ->
   0.25 : (position1'=winner) & (toss1'=1)
   + 0.25 : (position1'=loser) & (toss1'=2)
   + 0.5 : (toss1'=0); // 11.
  [] (nodeid1>nodeid0 & position1=contention
   & remaining_partner1=nodeid0 & toss0=1) ->
   (position1'=loser) & (toss0'=0); // 13.
  [] (nodeid1>nodeid0 & position1=contention
   & remaining_partner1=nodeid0 & toss0=2) ->
   (position1'=winner) & (toss0'=0); // 12.
  [] (nodeid1<nodeid2 & position1=contention
   & remaining_partner1=nodeid2 & toss1=0) ->
   0.25 : (position1'=winner) & (toss1'=1)
   + 0.25 : (position1'=loser) & (toss1'=2)
   + 0.5 : (toss1'=0); // 11.
  [] (nodeid1>nodeid2 & position1=contention
   & remaining_partner1=nodeid2 & toss2=1) ->
   (position1'=loser) & (toss2'=0); // 13.
  [] (nodeid1>nodeid2 & position1=contention
   & remaining_partner1=nodeid2 & toss2=2) ->
   (position1'=winner) & (toss2'=0); // 12.

// winner
  [] (position1=winner & remaining_partner1=nodeid0 & onezero=empty) ->
   (onezero'=be_my_parent) & (position1'=response); // 14.
  [] (position1=winner & remaining_partner1=nodeid2 & onetwo=empty) ->
   (onetwo'=be_my_parent) & (position1'=response); // 14.

// loser
  [] (position1=loser & remaining_partner1=nodeid0
   & zeroone=be_my_parent) ->
   (zeroone'=empty) & (position1'=child_handshake) & (adj1'=1)
   & (remaining_partner1'=0) & (no_of_requests1'=1) & (child1_0'=nodeid0)
   & (child1_1'=0) & (child1_2'=0); // 15.
  [] (position1=loser & remaining_partner1=nodeid2 & twoone=be_my_parent) ->
   (twoone'=empty) & (position1'=child_handshake) & (adj1'=1)
   & (remaining_partner1'=0) & (no_of_requests1'=1) & (child1_0'=nodeid2)
   & (child1_1'=0) & (child1_2'=0); // 15.

  [] (position1=become_child & onezero=empty & child1_0=nodeid2) ->
   (position1'=finish) & (onezero'=ack) & (no_of_requests1'=0)
   & (remaining_partner1'=0) & (adj1'=0) & (child1_0'=0)
   & (child1_1'=0) & (child1_2'=0); // 9.
  [] (position1=become_child & onetwo=empty & child1_0=nodeid0) ->
   (position1'=finish) & (onetwo'=ack) & (no_of_requests1'=0)
   & (remaining_partner1'=0) & (adj1'=0) & (child1_0'=0) & (child1_1'=0)
   & (child1_2'=0); // 9.

endmodule



module node2

  position2 : [start..finish] init start;
  child2_0 : [0..N] init 0;
  child2_1 : [0..N] init 0;
  child2_2 : [0..N] init 0;
  adj2 : [0..v2] init v2;
  remaining_partner2 : [0..N] init 0;
  no_of_requests2 : [0..N] init 0;

  [] (position2=start & onetwo=be_my_parent) ->
   (position2'=child_handshake) & (no_of_requests2'=1) & (onetwo'=empty)
   & (child2_0'=nodeid1); // 1.
  [] (position2=start & onetwo=empty) ->
   (position2'=child_handshake) & (no_of_requests2'=0)
   & (remaining_partner2'=nodeid1); // 4.

  [] (position2=child_handshake & adj2=v2 & remaining_partner2=0
   & twoone=empty) ->
   (position2'=parent_handshake) & (twoone'=be_my_child); // 2.
  [] (position2=child_handshake & adj2=v2 & remaining_partner2=nodeid1
   & twoone=empty) ->
   (position2'=parent_handshake) & (twoone'=be_my_parent); // 5.

  [] (position2=child_handshake & adj2=1 & no_of_requests2=1
   & child2_0=nodeid1 & twoone=empty) ->
   (position2'=parent_handshake) & (twoone'=be_my_child); // 16.
  [] (position2=parent_handshake & adj2=v2 & remaining_partner2=0
   & onetwo=ack) ->
   (position2'=finish) & (no_of_requests2'=0) & (elected'=nodeid2)
   & (adj2'=0)& (child2_0'=0)& (child2_1'=0) & (child2_2'=0)
   & (onetwo'=empty); // 3.
  [] (position2=parent_handshake & adj2=v2 & remaining_partner2=nodeid1) ->
   (position2'=handshakes_complete); // 6.

  [] (position2=parent_handshake & adj2=1 & no_of_requests2=1
   & child2_0=nodeid1 & onetwo=ack) ->
   (position2'=finish) & (onetwo'=empty) & (elected'=nodeid2) &
   (no_of_requests2'=0) & (child2_0'=0) & (adj2'=0); // 17.

  [] (position2=handshakes_complete & no_of_requests2=adj2-1) ->
   (position2'=response); // 7.

  [] (position2=response & remaining_partner2=nodeid1
   & onetwo=be_my_child) ->
   (position2'=become_child) & (remaining_partner2'=0) & (onetwo'=empty); // 8.
  [] (position2=response & remaining_partner2=nodeid1
   & onetwo=be_my_parent) ->
   (position2'=contention) & (onetwo'=empty); // 10.

// contention (probabilistic choice always made by nodeid with smallest id)
  [] (nodeid2<nodeid1 & position2=contention
   & remaining_partner2=nodeid1 & toss2=0) ->
   0.25 : (position2'=winner) & (toss2'=1)
   + 0.25 : (position2'=loser) & (toss2'=2)
   + 0.5 : (toss2'=0); // 11.
  [] (nodeid2>nodeid1 & position2=contention
   & remaining_partner2=nodeid1 & toss1=1) ->
   (position2'=loser) & (toss1'=0); // 13.
  [] (nodeid2>nodeid1 & position2=contention
   & remaining_partner2=nodeid1 & toss1=2) ->
   (position2'=winner) & (toss1'=0); // 12.

// winner
  [] (position2=winner & remaining_partner2=nodeid1 & twoone=empty) ->
   (twoone'=be_my_parent) & (position2'=response); // 14.

// loser
  [] (position2=loser & remaining_partner2=nodeid1 & onetwo=be_my_parent) ->
   (onetwo'=empty) & (position2'=child_handshake) & (adj2'=1)
   & (remaining_partner2'=0) & (no_of_requests2'=1) & (child2_0'=nodeid1)
   & (child2_1'=0) & (child2_2'=0); // 15.

  [] (position2=become_child & twoone=empty) ->
   (position2'=finish) & (twoone'=ack) & (no_of_requests2'=0)
   & (remaining_partner2'=0) & (adj2'=0) & (child2_0'=0)
   & (child2_1'=0) & (child2_2'=0); // 9.

endmodule
```

Appendix F

PRISM specification for Itai Rodeh protocol

```
nondeterministic

const int ACTIVE = 0;
const int PASSIVE = 1;
const int LEADER = 2;
const int FINISH = 3;

global leader : [0..N] init 0;

const int id0=1;
const int id1=2;
const int id2=3;

const int N=3; // number of processes


module process0

// COUNTER
```

```
c0 : [0..N-1];

// STATES
s0 : [ACTIVE..FINISH];

// PREFERENCE
p0 : [0..2] init 2;

// VARIABLES FOR SENDING AND RECEIVING
receive0 : [0..2];
// 0 not received anything
// 1 received choice
// 2 received counter

sent0 : [0..2];
// 0 not send anything
// 1 sent choice
// 2 sent counter

// pick value
[] (s0=ACTIVE & p0=2) -> 0.5 : (p0'=0) + 0.5 : (p0'=1);

// send preference
[p_0_1snd] (s0=ACTIVE) & (p0<2) & (sent0=0) -> (sent0'=1);

// receive preference
// stay active
[p_2_0rcv] (s0=ACTIVE) & (p0<2) & (receive0=0)
& ((p0=1 & ch_2_0_1=0) | (p0=1 & ch_2_0_1=1) | (p0=0 & ch_2_0_1=0)) ->
(receive0'=1);
// become inactive
[p_2_0rcv] (s0=ACTIVE) & (p0<2) & (receive0=0) & (p0=0) & (ch_2_0_1=1) ->
(s0'=PASSIVE) & (receive0'=1);
```

```
// send counter (already sent preference)
// not received counter yet
[c_0_1snd] (s0=ACTIVE) & (p0<2) & (sent0=1) & (receive0=1) ->
(sent0'=2);
// received counter (pick again)
[c_0_1snd] (s0=ACTIVE) & (p0<2) & (sent0=1) & (receive0=2) ->
(p0'=2) & (c0'=0) & (sent0'=0) & (receive0'=0);

// receive counter and not sent yet
// (note in this case do not pass on as will send own counter)
[c_2_0rcv] (s0=ACTIVE) & (p0<2) & (receive0=1) & (sent0<2) ->
(receive0'=2);

// receive counter and sent counter
// only active process (decide)
[c_2_0rcv] (s0=ACTIVE) & (p0<2)  & (receive0=1) & (sent0=2) & (ch_2_0_1=N-1) ->
(s0'=LEADER) & (p0'=2) & (c0'=0) & (sent0'=0) & (receive0'=0);
// other active process (pick again)
[c_2_0rcv] (s0=ACTIVE) & (p0<2)  & (receive0=1) & (sent0=2) & (ch_2_0_1<N-1) ->
(p0'=2) & (c0'=0) & (sent0'=0) & (receive0'=0);

// send preference (must have received preference)
[p_0_1snd] (s0=PASSIVE) & (p0<2) & (receive0>0) & (sent0=0) ->
(sent0'=1);

// send counter (must have received counter first) and can now reset
[c_0_1snd] (s0=PASSIVE) & (p0<2) & (receive0=2) & (sent0=1) ->
(s0'=PASSIVE) & (p0'=2) & (c0'=0) & (sent0'=0) & (receive0'=0);

// receive preference
[p_2_0rcv] (s0=PASSIVE) & (p0=2) & (receive0=0) & (ch_2_0_1<2) ->
(p0'=ch_2_0_1) & (receive0'=1);
```

```
// receive counter
[c_2_0rcv] (s0=PASSIVE) & (p0<2) & (receive0=1) & (ch_2_0_1<N-1) ->
(c0'=ch_2_0_1+1) & (receive0'=2);

// store leader
[] (s0=LEADER) ->  (leader'=id0) & (s0'=FINISH);

  // finished (loop)
[done] (s0=FINISH) ->  (s0'=s0);
[done] (s0=PASSIVE) -> (s0'=s0);

endmodule



module chan_0_1

ch_0_1_1 : [0..N] init N;
ch_0_1_2 : [0..N] init N;
ch_0_1_3 : [0..N] init N;

// buffer preference
[p_0_1snd] (ch_0_1_1=N) -> (ch_0_1_1'=p0);
[p_0_1snd] (ch_0_1_1!=N& ch_0_1_2=N)-> (ch_0_1_2'=p0);
[p_0_1snd] (ch_0_1_1!=N& ch_0_1_2!=N& ch_0_1_3=N)-> (ch_0_1_3'=p0);

// buffer counter
[c_0_1snd] (ch_0_1_1=N) -> (ch_0_1_1'=c0);
[c_0_1snd] (ch_0_1_1!=N& ch_0_1_2=N)-> (ch_0_1_2'=c0);
[c_0_1snd] (ch_0_1_1!=N& ch_0_1_2!=N& ch_0_1_3=N)-> (ch_0_1_3'=c0);

// transmit preference
[p_0_1rcv] (ch_0_1_1!=N) ->
```

```
(ch_0_1_1'=ch_0_1_2) & (ch_0_1_2'=ch_0_1_3) & (ch_0_1_3'=N);

//transmit counter
[c_0_1rcv] (ch_0_1_1!=N) ->
(ch_0_1_1'=ch_0_1_2) & (ch_0_1_2'=ch_0_1_3) & (ch_0_1_3'=N);

endmodule

module process1=process0[
id0=id1,s0=s1,p0=p1,c0=c1,sent0=sent1,receive0=receive1,
p_0_1snd=p_1_2snd,p_2_0rcv=p_0_1rcv,c_0_1snd=c_1_2snd,
c_2_0rcv=c_0_1rcv,ch_2_0_1=ch_0_1_1] endmodule

module chan_1_2=chan_0_1[
p0=p1,c0=c1,p_0_1snd=p_1_2snd,p_0_1rcv=p_1_2rcv,
c_0_1snd=c_1_2snd,c_0_1rcv=c_1_2rcv,
ch_0_1_1=ch_1_2_1,ch_0_1_2=ch_1_2_2,ch_0_1_3=ch_1_2_3
] endmodule

module process2=process0[
id0=id2,s0=s2,p0=p2,c0=c2,sent0=sent2,receive0=receive2,
p_0_1snd=p_2_0snd,p_2_0rcv=p_1_2rcv,c_0_1snd=c_2_0snd,
c_2_0rcv=c_1_2rcv,ch_2_0_1=ch_1_2_1] endmodule

module chan_2_0=chan_0_1[
p0=p2,c0=c2,p_0_1snd=p_2_0snd,p_0_1rcv=p_2_0rcv,
c_0_1snd=c_2_0snd,c_0_1rcv=c_2_0rcv,ch_0_1_1=ch_2_0_1,
ch_0_1_2=ch_2_0_2,ch_0_1_3=ch_2_0_3] endmodule
```

Appendix G

PRISM ‘Individuals’ Specification for SIR Protocol

```
probabilistic

    const double B=1;     // probability of infection
    const double Q=1;     // rate of removal

    const int Susceptible=0;
    const int Infective=1;
    const int Removed=2;

module site1

   state1 : [Susceptible..Removed] init Susceptible;

   [] (state1=Infective) ->
     Q : (state1'=Removed) + (1-Q) : (state1'=state1);

   // site is susceptible and is contacted by an infected site
   [] state1=Susceptible & state2=Infective ->
     B : (state1'=Infective) + (1-B) : (state1'=state1);
   [] state1=Susceptible & state3=Infective ->
     B : (state1'=Infective) + (1-B) : (state1'=state1);



endmodule

module site2 =site1[
state1=state2,deliver1=deliver2,state2=state1
] endmodule

module site3
   state3 : [Infective..Removed] init Infective;

   [] (state3=Infective)
    -> Q : (state3'=Removed) + (1-Q) : (state3'=state3);
endmodule
```

Appendix H

PRISM ‘Population’ Specification for SIR Protocol

```
// population model derived from individual SIR model

probabilistic          // DTMC model

    const int N=3;     // number of sites

    const double B=1;  // probability of infection
    const double Q=1;  // protbability of removal


module population

    s : [0..N] init N-1;    // number of susceptibles
    i : [0..N] init 1;      // number of infectives

    [] (i>0 & s>0) -> (B*s/(s+1)) : (s'=s-1) & (i'=i+1)
                    + (Q/(s+1)) : (i'=i-1)
                    + (1-((B*s+R)/(s+1))) : (s'=s);
    [] (i>0 & s=0) -> (Q/(s+1)) : (i'=i-1)
```

```
                        + (1-((B*s+Q)/(s+1))) : (s'=s);
    [] (i=0)           -> 1 : (i'=i);

endmodule
```