MODEL CHECKING OF PROBABILISTIC TIMED AND HYBRID SYSTEMS

by

JEREMY JAMES SPROSTON

A thesis submitted to the Faculty of Science
of The University of Birmingham
for the Degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
Faculty of Science
University of Birmingham
October 2000

Abstract

The development of digital technology embedded in real-life environments has lead to increased interest in formal techniques for real-time and hybrid systems. Traditionally, models for these systems, such as timed and hybrid automata, feature purely nondeterministic choice between enabled transitions, abstracting potentially significant information concerning the probability of behaviours. For example, the fact that a component malfunction occurs with a certain probability may be of vital importance when verifying the reliability of a system. Similarly, the possibility that a real-time system fails to meet a certain deadline with a low probability may be acceptable in certain applications, such as telecommunications or multimedia protocols.

Therefore, we present probabilistic timed and hybrid automata as modelling formalism for such systems, and give model checking methods for the automatic verification of their correctness against properties which refer directly to probability. For probabilistic timed automata, we present a method to obtain a finitary partition of the infinite state space of the model, which can then be verified against a probabilistic temporal logic. Techniques for the translation of subclasses of probabilistic hybrid automata into probabilistic timed automata are also given. Finally, we introduce methods using based on search through the state space of probabilistic timed and hybrid automata, which can be used for the verification of probabilistic reachability properties.

Acknowledgements

I would like to thank my supervisor, Marta Kwiatkowska, for her exceptional support throughout my time as a PhD student. Her enthusiasm for this topic of research and her faith in my abilities were invaluable. I would also like to thank Gethin Norman for his patience and endless technical support, and Roberto Segala for his insight and advice, particularly with regard to Appendix A. Also deserving of thanks are Dave Parker and Mark Ryan.

Finally, I would like to thank my parents, my sister and Viviana for their support.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

The ubiquity of examples of discrete technology embedded into real-life environments has lead to a growth of interest in techniques to increase our confidence in their performance and reliability. That such a system satisfies its specified requirements is desirable both from the point of view of the consumer and the manufacturer of a product. For example, the consumer's interest in the reliability of the braking system of a state-of-the-art BMW is clear, as is the producer's desire to maintain a relationship of trust with its informed customers. Similarly, it is in the interests of both parties for the manufacturing process of the product to be as cost effective as possible. This interplay prompts a demand for inexpensive and convenient ways to ensure that designs of complex embedded computer systems are of a high quality. The availability of conventional computing technology has prompted interest in automatic, *algorithmic* techniques with which such guarantees may be obtained by engineers and system designers.

There are two aspects to such a notion of *verification* of systems. Firstly, appropriate formalisms for the *description* of systems and their requirement specifications must be developed. Observe that the type of systems described above features interaction between two types of components: discrete and continuous. As such, these systems are called *hybrid systems*. For example, the computer embedded into a car is discrete, as it is digital in nature, whereas the car itself operates in a continuous environment, in which aspects such as speed and temperature vary continuously. Other examples of hybrid systems include aircraft, robots, medical equipment and washing machines. The computer science community has developed a formalism called *hybrid automata* [ACHH93, NOSY93], in which the discrete components of a hybrid system are represented by a finite automaton, and the continuous components are represented by a finite set of real-valued variables. Interaction between these components is represented by conditions on the variables that label the vertices and edges of the automaton, and include conditions which enable edges for nondeterministic choice, and differential equations or inequalities which specify how the variables change as time elapses. The semantics of a hybrid automaton is given by an infinite-state labelled transition system, which performs two types of transitions: discrete, in which no time elapses, but a move across an edge of the automaton is performed; and continuous, in which the current vertex of the automaton remains the same, but the values of the variables change according to the differential equation or inequality labelling that vertex. Similarly, a formal way of specifying requirements on both the continuous and discrete components must also be used. Computer scientists have extended the formalism of temporal logic [Pnu77] for this purpose.

The second requirement for a verification technique for systems is, naturally, an *analysis method*. Such a method may take the form of rules with which a manual proof of correctness, which establishes that the formal model of the system satisfies its specification, can be carried out. Alternatively, and as mentioned above, the method may take the form of an algorithmic analysis. It is this latter category which will be the focus in this thesis. In particular, we will concentrate on the technique of *model checking*, which performs an automatic exploration of portions of the state space of the system that are relevant for the satisfaction or refutation of a temporal logic formula [CE81, QS82]. Model checking algorithms have been implemented in a number of tools, such as Smv [McM93] and Spin [Hol97], and have been particularly successful in the verification of finite-state, discrete systems, such as hardware and protocols [BCM+92, CK96]. Furthermore, model checking has been extended to the case of hybrid automata in [ACHH93, NOSY93, AHH96], which reduce the infinite state space of the model in question to a finitary representation with enough granularity to answer the verification problem. Such algorithms form the basis of the tool HyTech [HHW97]. A number of case studies of hybrid system verification using HyTech have been presented [HW95, HW96], although some limitations of this tool have been highlighted, in particular in a case study carried out in partnership with Bmw [SMF97]. The tool HyTech is currently under redevelopment [HHMW00].

An interesting subclass of hybrid systems is that of *real-time systems*. These are systems for which quantitative timing issues are of vital importance, such as a system subject to the constraint given by "an event response occurs within 5 time units after the event request". Real-time systems can be regarded as hybrid systems upon consideration of the fact that their continuous component is isomorphic to time (as opposed to hybrid systems, in which the continuous components are defined as having a relationship with time according to differential equations and inequalities). Similarly, the class of *timed automata* [AD94], which are appropriate for the modelling of real-time systems, are a subclass of hybrid automata for which all variables increase at the same rate as real-time. Model checking algorithms for timed automata have been presented in [AD94, ACD93, HNSY94, BTY97], and have been implemented in the tools Uppaal

[LPY97a] and Kronos [Yov97]. Many successful applications of these tools to real-life case studies have been presented, including [MY96, DKRT97, LPY98, TY98, Tri99a].

The transitions of a timed or hybrid automaton are nondeterministic in nature; that is, in any given state of such a model, there is a nondeterministic choice between the set of available transitions. However, this means that we treat system behaviours, which are represented as sequences of transitions, in a uniform manner. For example, consider two sets of behaviours, one in which the brakes of a Bmw fail with high probability, perhaps because this set corresponds to adverse weather or road conditions, the other in which they fail with low probability, which may correspond to more favourable environmental conditions. Previous approaches to hybrid system analysis only distinguish between the cases in which none, all, or some of the behaviours in such a set correspond to the brakes failing, and therefore both of the sets described here (in which *some* of the paths considered correspond to brake failure) would be treated in the same manner. However, the system designer may regard failure with a low probability to be acceptable, perhaps because to guarantee a higher standard of reliability may be substantially more costly. Therefore, the association of notion of probability or likelihood with behaviour may be appropriate for the verification of certain hybrid systems. Naturally, these systems are called *probabilistic hybrid systems.*

The concept of viewing the transitions of a system in terms of probabilistic choice has many precedents in computer science. To understand the way in which the modelling of probability may be advantageous within the context of hybrid systems, we consider general classes of system for which estimates of the likelihood of events may be obtained in a natural manner, and for which such estimates may be highly significant with respect to system correctness. Firstly, some systems operate according to *randomised algorithms*, which specify that certain transitions of computing devices are determined explicitly according to probabilistic choice (such as that obtained by electronic coin flipping). Randomisation can be used, for example, to break the symmetry between identical processes. Secondly, certain systems, for which the occurrence of a fault in one of its components does not necessarily entail incorrectness (for example, if a "fault handling" mechanism an an explicit feature of the system's design), are classed as being *fault-tolerant.* In such cases, the frequency of occurrence of a fault, and the likelihood of a return to normal, non-faulty operation, may be associated with statistical estimates, from which probabilities of transitions may be derived. A third class of system for which estimates of likelihood are both natural and potentially significant are those for which probabilistic information may be associated with the *environment* in which the system is operating. For example, the rate of arrival of users at a network may be associated with a statistical distribution. We note that all three ways in which probability may arise in the formal description of computer system also apply to real-time and hybrid systems. An example of a real-time system operating according to a randomised algorithm is the IEEE1394 (FireWire) root contention protocol (as described in [SV99, SS00, D'A99a]). An example of a hybrid system for which probabilities may be associated with component failures include the steam boiler described in [Abr96]. Finally, a computer controller embedded in a vehicle operating in an environment for which statistical measures concerning the environment (for example, weather conditions) is an example of a hybrid system for which probability may be associated with its continuous environment [PLNS99, HLPS99].

Given that we have a probabilistic description of a system, we may be interested in the probability with which the system satisfies a certain property. For example, in a system employing a randomised algorithm, we may require a property (such as the election of a leader, for example) to be exhibited with probability 1. In a fault-tolerant system, the probability of the occurrence of a costly series of failures may be required to be below a certain, low probability. Finally, we may require a system embedded in a probabilistic environment to satisfy a property (relating to the throughput of a network, for example), with a given probability or greater. The introduction of such reference to probability in both the system description and the requirement specification has prompted the development of *probabilistic model checking*. As in non-probabilistic, traditional model checking, the aim of this technique is to automatically verify that the system satisfies its specification; therefore, although the system designer supplies the probability values relating to the system and its requirement specification, the probabilistic model checking algorithm establishes the satisfaction of the requirement without the designer needing to know details concerning the way in which the probabilities expressed in the system and the requirement interact.

The remit of this thesis is to extend the model of hybrid automata with probability to provide a formal model for probabilistic hybrid systems, and to derive an associated model checking technique. The feasibility of such an approach is suggested not only by the success of methods for the description of hybrid systems, but also those concerning *finite-state, probabilistic systems.* Such systems are traditionally modelled by computer scientists by finite-state Markov chains or Markov decision processes. First consider Markov chains; for any state, the choice of which state to make a transition to is made according to a probability distribution. Markov decision processes are a generalisation of Markov chains in which a set of probability distributions is associated with each state. A transition of a Markov decision process is performed first by choosing a distribution from such a set nondeterministically, and then making a probabilistic choice as to which state to go to according to the chosen distribution. Both Markov chains and Markov decision processes have been subject to scrutiny by the model checking community, and algorithms have been developed for automatically verifying whether such models satisfy temporal logic properties. Such properties may be represented using traditional forms of temporal logic; then a formula is interpreted as being valid if the probability of the behaviours of the model which satisfy the formula is 1 [Var85, CY95]. Alternatively, temporal logics can be augmented with operators that can express bounds on the probability of the satisfaction of their subformulae [HJ94, BdA95, dA97a, BK98, Bai98]. Then such an operator resolves to true if the probability of the set of behaviours which satisfy its subformulae also satisfies the bound. Furthermore, recent research has concerned the verification of *probabilistic timed systems*, such as continuous-time Markov chains, for which there exist both the notions of real-time and probability [ACD91a, ACD91b, dA97a, dA98b, BKH99]. Model checking tools for untimed Markov decision processes include Prism [dAKN^{+}00], and $\mathsf{E} \vdash \mathsf{MC}^2$ [HKMS00] for continuous-time Markov chains.

1.2 Outline of the thesis

The thesis comprises two main parts. The first concerns the definition of *probabilistic timed automata*, which constitute a formal model for probabilistic timed systems, and the presentation of two analysis methods. The second main section introduces *probabilistic rectangular automata*, a subclass of the probabilistic hybrid automata presented in [Spr00], and shows how model checking may be performed on such a class. Throughout the thesis, our aim is to provide model checking methods for probabilistic temporal logics for which there is an operator which can specify the required bound on the satisfaction of subformulae, rather than the simpler case in which we require satisfaction of subformulae with probability 1.

Chapter 2 presents an overview of the related literature in this field, with particular emphasis being given to model checking methods for timed and hybrid automata, and also those for Markov chains and Markov decision processes. Chapter 3 introduces some preliminary technical concepts, including a model which is similar to Markov decision processes, notions of state preorders and equivalences for these models, a probabilistic temporal logic, and a means for reasoning about dense state spaces.

Chapter 4 introduces the model of probabilistic timed automata, giving both their syntax and semantics. Intuitively, if a classical timed automaton can be regarded as a *finite automaton* equipped with a set of variables which increase continuously at the same rate as real-time, then a probabilistic timed automaton can be thought of as a *Markov decision process* equipped with such variables. This chapter also presents a concept of realisability of their behaviours, which is derived naturally from notions of discrete and time progress presented elsewhere in the literature, and suggests well-formedness requirements for guaranteeing that all behaviours of the model are realisable.

Chapter 5 presents a timed, probabilistic logic for the specification of properties of probabilistic timed automata, and a model checking method based on the well-known region construction of [AD94]. That is, we partition the entire state space of the model according to an equivalence which preserves formulae of the logic; this partition can then be used to define a finitary Markov decision process, which can subsequently be subject to traditional, untimed probabilistic verification techniques.

In Chapter 6, we introduce an alternative method for the verification of probabilistic timed automata, which is based on forward traversal of a portion of the model's state space in order to verify probabilistic reachability properties (essentially a subclass of the properties expressible in the logic of Chapter 5). These properties can be used to state requirements such as "with probability 0.99 of greater, the system enters a target state within 5 time units". The method proceeds by iteration of a successor operation on state sets to obtain a part of the state space which is reachable with positive probability. Then the state sets generated by this process are represented by states of a finite Markov decision process, which can be analysed using the maximal probability reachability algorithm of [BdA95, dA97a, dA99b].

Chapter 7 introduces the model of probabilistic rectangular automata, which can be regarded as a Markov decision process equipped with variables which change continuously according to convex, integer bounded differential inclusions. We also suggest how this model may be used to formalise a possibly faulty steam boiler, with our approach based on that of [HW96, MMT98].

Chapters 8 and 9 suggest model checking methods for probabilistic rectangular automata. Firstly, methods for the translation of subclasses of this model to that of probabilistic timed automata are introduced in Chapter 8. These methods preserve the properties of the probabilistic temporal logic presented in Chapter 3. Secondly, Chapter 9 features a forward traversal algorithm for probabilistic rectangular automata, which is based on that of Chapter 6. We also apply this algorithm to the faulty steam boiler of Chapter 7. Finally, Chapter 10 evaluates the thesis, and suggests further directions of research.

Note that material from Chapters 4 and 5 appeared in [KNSS99, KNSS00a], the latter of which also included results from Chapter 6. The author arrived at the definition of probabilistic timed automata in Chapter 4 following a suggestion by Roberto Segala to simplify a model to that which appeared later in [KNSS00b]. The notion of divergence was defined by all four authors of [KNSS99, KNSS00a], although the static conditions for discrete progress were the work of the author of this thesis. Furthermore, the author was a major contributor to the details of the associated model checking method presented in Chapter 5. In particular, the author developed the use of region equivalence as a method to obtain a finite-state system from a probabilistic timed automaton; the associated proofs of Lemma 5.4.7, Lemma 5.4.8, Lemma 5.4.9, and Proposition 5.5.3 were mainly co-authored with Gethin Norman. The material contained in Chapter 6 was instigated and developed by the author, subject to detailed checking and revisions by the other three authors of [KNSS00a]. A number of results (more precisely, Lemma 6.6.1, Lemma 6.6.4, Lemma 6.6.5 and Proposition 6.6.6) were obtained in collaboration with Gethin Norman.

Material from Chapters 7 and 8 appeared in [Spr99, Spr00].

Chapter 2

Background

2.1 Introduction

This chapter presents an overview of the model checking techniques that have been developed for timed automata, hybrid automata, and probabilistic systems. The following survey is not meant to be exhaustive, but instead is intended to summarise the major results in the above topics, and other results that are particularly relevant for this thesis. Section 2.2 covers the field of model checking for discrete, non-probabilistic systems, while Section 2.3 and Section 2.4 introduce timed and hybrid automata, respectively, and surveys their application to the modelling and verification of real-life systems. Probabilistic systems, whether including a notion of real-time or not, are focussed on in Section 2.5.

2.2 Discrete systems

2.2.1 Temporal logic

One of the dominant languages for formal reasoning about computer systems is that of *temporal logic*. This is a type of modal logic in which the modalities $\Box$ (inevitability) and $\Diamond$ (possibility) are interpreted in a temporal context, so that $\Box$ means "at all time points in the future", and $\Diamond$ means "at some time point in the future". Alternatively, both of these modalities may be represented as a single operator, called "until"; for example, if p is an atomic proposition, then "$\texttt{true}$ until p" is equivalent to $\Diamond p$, and then $\neg\Diamond\neg p$ is equivalent to $\Box p$. Although temporal logic has its foundations in philosophy, its use to computer scientists was highlighted by Pnueli [Pnu77].

Temporal logic has been widely used to define manual proof methods for the correctness of computer systems, such as those given in [MP92], which also form the basis of semi-automated theorem proving techniques, such as those implemented in the tool STeP [BBC$^+$96]; however, we will concentrate on the application of this type of logic to model checking. Note that temporal logic is interpreted over structures such as Kripke structures, labelled transition systems, or state machines; the minor distinctions between these are not relevant here, and we will henceforth describe structures such as these as labelled transition systems, abbreviated to LTSs.

A system can also be represented by the set of its possible *behaviours* or *computations*. The connection between such a representation and that of a LTS encoding can be understood on consideration of the fact that traversing successively the transitions of the LTS results in a sequence (of states, of events, or of both) which denotes a particular behaviour of the system. Such a sequence is generally called a *path*. The possible behaviours of the system from a particular state can be represented in two ways. On one hand, they could be represented (in a monolithic way) as a set of paths which commence in the relevant state. Alternatively, they could be represented in a more structured manner as a *computation tree*, in which the nodes correspond to states, and the arcs represent transitions. More precisely, the root node is the relevant start state of the paths, and the children of any node are the set of successor states (obtained from the transition relation) of the state which the node represents. Then a particular path of the system is obtained by traversing a branch of the computation tree. We then say that a temporal logic is *linear* if the validity of its formulae are interpreted over individual paths, and *branching* if its formulae are interpreted over computation trees.

Two temporal logics have received much attention, particularly in the context of automatic verification methods such as model checking: namely the linear-time logic LTL, which has its basis in the logic proposed by [Pnu77], and the branching-time logic CTL [CE81]. Observe that the latter is obtained by adding *path quantifiers* to the syntax of temporal logic, following the precedent of Ben-Ari, Manna and Pnueli [BMP83]; that is, a subformula φ which expresses properties of paths is nested directly within a path quantifier, which can either take the form of $\forall$ or $\exists$. We say that $\forall$ is the *universal* path quantifier, and $\exists$ is an *existential* path quantifier, where the CTL formula $\forall\varphi$ ($\exists\varphi$) is true in a particular state if all (some, respectively) paths leaving the state satisfy φ. The expressive powers of the logics LTL and CTL are incomparable (see [CGL94]). The branching-time logic CTL*, which subsumes LTL and CTL, was introduced in [EH86], and permits arbitrary nesting of formulae expressing properties of paths within path quantifiers. The state formulae appearing as subformulae of these path properties may themselves be CTL* formulae.

We briefly note that LTSs can be equipped with a notion of *fairness* (see [MP92]). That is, if a LTS represents the behaviour of a finite number of interacting components, then a path in which the component takes only finitely many transitions could be regarded as being unrealistic; informally speaking, the computation is unfair to the component in question. This notion has particular significance on consideration of the fact that we may wish to evaluate temporal logic formulae only over fair paths, or over fair transition systems [MP92]. We note that alternative interpretations of fairness exist in the literature, and refer the reader to [MP92] for a survey.

2.2.2 Model checking

The phrase "model checking" was coined by Emerson and Clarke to describe an algorithmic method for verifying that a system satisfies a specification [CE81]. The pioneering work of these authors, and of the independent work of Quielle and Sifakis [QS82], considered the case in which the system model is represented as a finite-state LTS, and the specification is represented as a CTL formula. The aim of their model checking algorithm is to label states with the subformulae of a CTL formula which they satisfy. For a subformula with a path quantifier and a temporal modality as its outermost operator, the labelling is propagated by backwards traversal of the LTS's transition relation. The method of Clarke and Emerson accommodated an appropriate notion of fairness; as this notion could not be encoded as a CTL formula, the semantics of CTL, and the basic model checking procedure, are adjusted accordingly. The method was later improved in [CES86], and applied to real-life examples in [DC86, BCDM86].

Tableau and automata-theoretic approaches

Methods tailored exclusively for the automatic verification of linear-time properties, such as those expressed in the logic LTL, also appeared in the 1980s. As summarised in the survey paper [CGL94], the approach of Lichtenstein and Pnueli was to construct a "tableau" LTS representation of the LTL formula in question, which is composed with the model of the system to result in a third LTS. Finally, appropriate analysis is then performed on this third system. Similarly, Vardi and Wolper take an automata-theoretic approach, in which both the system model and the linear-time temporal logic formula are represented by finite automata over infinite sequences [VW86]. Model checking then proceeds by constructing the product automaton of the automaton which represents the system (for which all states are accepting), and the automaton representing the *negation* of the LTL formula in question, and then testing the resulting product automaton for emptiness. As noted by [Var96], this technique is in the category of the *language inclusion* methods developed by Kurshan, in which verification is performed by checking whether the language (the set of infinite computations) of the system is included in that of the specification. This automata-theoretic approach to model checking was extended from linear-time temporal logic to the branching-time logics CTL and CTL* in [KVW00]. Automata-theoretic techniques for linear-time temporal logic specifications have been implemented by Holzmann in the model checker Spin [Hol97].

Symbolic model checking

Unfortunately, model checking techniques are subject to the so-called *state space explosion problem*, which refers to the fact that the global state space of a system increases exponentially in relation to the number of its parallel components, which is a significant obstacle to the application of such methods to real-life systems. A major breakthrough came with the work of McMillan [BCM$^+$92, McM93], which proposed the *symbolic* representation of state sets and the transition relation of the system in terms of boolean formulae, which could then be encoded into compact, efficient and canonical data structures called Binary Decision Diagrams (BDDs) [Bry86]. Hence, the term *symbolic model checking*, which is used to describe model checking techniques in which an implicit, symbolic representation of the system is stored and manipulated, was coined. Although the use of BDDs could potentially be no better than the previous, enumerative techniques, the implementation of this data structure in the algorithm of the model checking tool Smv (Symbolic Model Verifier), developed by McMillan, resulted in the successful verification of systems that were out of the scope of previous approaches [BCM$^+$92]. Symbolic model checking techniques were also implemented in the tool Cospan, developed by Kurshan (see [CK96]).

Equivalences and preorders

We now consider how certain developments elsewhere in the theory of concurrency have been applied to model checking. One of the many rich theories concerning the models of computation studied in the field of process algebra is that of *preorders* or *equivalences* between system states. In particular, we consider the preorder of *simulation* and the equivalence relation of *bisimulation* (see [Mil89]). In the context of model checking, such relations provide a way to reduce the state space in size before verification is performed. First, we focus on bisimulation: it is shown in [BCG88] that bisimilar states of LTSs satisfy the same formulae of CTL* (and therefore CTL and LTL). Now consider a LTS, the state space of which contains a state for each bisimulation equivalence class of the original system, and the transition relation of which is obtained in a natural way: there exists a transition from one state s of the new system to another s' if there exists a transition from a state in the equivalence class represented by s to a state in the equivalence class represented by s'. Such a system is called a *quotient*. Then, as the quotient system satisfies a CTL* property if and only if the original LTS satisfies the property, model checking can be performed on the quotient to obtain the desired result for the original system. Algorithms to compute bisimulation equivalence classes of systems are based on a "partition refinement" concept [KS83, PT87].

We now consider simulation. It is shown in [GL94] that a LTS $\mathcal{S}_1$ that simulates another $\mathcal{S}_2$ is such that, if $\mathcal{S}_1$ satisfies a property of the logic $\forall$CTL*, then $\mathcal{S}_2$ satisfies the property, where $\forall$CTL* is a fragment of CTL* which allows only universal quantification over paths. This approach has connections with that of *abstraction* applied to model checking [BBLS92, CGL94, DGG97], in which similar property preservation results are obtained.

2.3 Timed automata

We now turn our attention to the way in which the verification methods of the previous section can be expanded to reason about *real-time systems.* We use this term to describe systems for which *quantitative timing* issues are important; that is, it is not enough (in terms of system description and analysis) to say that one event happens after another, and instead we must say that an event occurs a certain amount of time after another. In this section, we give an overview of *timed automata* [AD94], which is one of the most successful formalisms for the description of real-time systems within the theoretical computer science community, and which underlies the formalism for probabilistic real-time systems used in this thesis.

2.3.1 Introduction to timed automata

The model of timed automata is primarily inspired by the model for real-time systems of Dill [Dil89], which introduced the idea that events of the system could only occur within certain, predefined time bounds. Informally speaking, a timed automaton is a finite automaton plus a finite set of real-valued variables called *clocks.* Some or all of these clocks may be *reset* to 0 on any discrete transition of the automaton. While control of the timed automaton remains in a given automaton state, called a *location*, the values of all of the clocks increase at the same rate as real-time; therefore, it can be seen that a clock's value corresponds to the amount of time elapsed since it was last reset. Clocks can be reset at different times, and there may be no lower bound on the differences between their values.

To specify the interaction between the discrete part of the timed automaton, represented by the locations and edges of the finite automaton, and the continuous part, represented by the set of clocks, we introduce a notion of *clock constraints.* These will take the form of boolean combinations of formulae of the form $x \sim c$, where x is a clock, $\sim \in \{<, \leq, \geq, >\}$ is a comparison operator, and $c \in \mathbb{N}$ is a natural number. Each of the discrete transitions of the timed automaton is labelled with a clock constraint known as an *enabling condition.* A discrete transition is said to be *enabled* if its associated constraint is satisfied by the current clock values. This feature allows us to represent a variety of timing properties; for example, a bound can be imposed on the time between the occurrence of two transitions if a certain clock is reset when the first transition is made, and then an appropriate enabling condition that involves this clock is associated with the second transition.

Timed automata are then obtained by the identification of final locations, which allows the definition of acceptance conditions. The timed automaton can then be described as accepting a language of *timed words*, which take the form of infinite sequences of events labelled by the time of their occurrence. In addition to the natural requirement that the time point of an event should be at least that of the directly preceding event, Alur and Dill also require that the timed word exhibits *progress*, which stipulates that time increases beyond any finite bound, or, equivalently, that it does not converge.

Observe that a state of a timed automaton is a pair comprising of the location in which control currently resides, and a *valuation* of all of the clocks of the model, which either takes the form of a vector of real-values or a function from the set of clocks to the reals. Hence, the state space of timed automata is infinite.

2.3.2 Early results for timed automata

A number of model checking and decidability results were initially obtained by Alur and Dill using this automata-theoretic approach to real-time. Firstly, it is shown in [AD94] that the emptiness problem for timed automata is decidable. This result is obtained by the definition of an appropriate, *finite* equivalence relation on the infinite state space of a timed automaton, called *region equivalence.* Region equivalence is defined such that two states are equivalent if they agree on:

1. their location component;
2. the integral parts of their clock values; and
3. the ordering of the fractional parts of all of their values.

The significance of points 2 and 3 is as follows: point 2 implies that region equivalent states can select the same discrete transitions for choice at the present time, whereas point 3 implies that region equivalent states will enable the same discrete transitions as time passes. Region equivalence can then be used to define a finite automaton called the *region graph*, the states of which represent region equivalence classes, and the transitions of which are obtained from the timed automaton in a natural manner (they either correspond to the elapse of time or a discrete transition between locations of the timed automaton). Then the emptiness problem for the timed language of the timed automaton can be reduced to the emptiness problem of the untimed language of the region graph.

It is also shown in [AD94] that the use of clock constraints of the form $x - y \sim c$, where x, y are clocks, $\sim \in \{<, \leq, \geq, >\}$ and $c \in \mathbb{N}$, in addition to those of the form $x \sim c$, does not affect the decidability of the above problems. We henceforth refer to the set of clock constraints of the form $x \sim c$ only as *rectangular*, and those involving both types of constraint as *triangular* (this terminology is taken from [Hen96]; we note that [BDFP00a] calls rectangular clock constraints *diagonal-free*). Alur and Dill show that the emptiness problem for timed automata involving clock constraints such as $x \sim y + z$, where z is also a clock, is undecidable.

2.3.3 Model checking for timed automata

The introduction of timed automata was accompanied by an automata-theoretic model checking method for timed automata against quantitative, linear-time properties specified as deterministic timed automata [AD94, AD96]. Another early model checking approach to timed automata against TCTL (Timed Computation Tree Logic) is presented in [ACD93]. The logic TCTL features *bounds* associated with its "until" temporal operator; that is, the formula $\phi_1 \mathcal{U}_{[2,4]} \phi_2$ is interpreted as specifying that ϕ_2 is true between 2 and 4 units in the future, and ϕ_1 is true continuously until that time. As with the decidability of emptiness result discussed above, the algorithm proceeds by reducing the model checking problem on the infinite state space of the timed automaton to the associated problem on the finite region graph. The graph is equipped with a notion of fairness to ensure that only paths which exhibit progress are considered.

Unfortunately, the practical applicability of these model checking methods are limited by the fact that the size of the region graph grows exponentially not only with the number of parallel timed automaton components, but also with the total number of clocks and the largest natural constant used in the description of any such component; therefore, the usual "state space explosion" problem witnessed in the verification of finite-state systems is compounded. The problem provoked a wealth of techniques to alleviate this bottleneck. One of the first was that of [HNSY94], which suggested a *symbolic* method for the verification of timed automata. Of course, the region graph is symbolic in the sense that a number of states is encoded into a single region; however, it is convenient to regard the region graph an an *enumerative* method, with symbolic methods applied to the region graph being regarded as *symbolic* in this context. The approach of Henzinger et al. was to represent state sets as conjunctions of clock constraints and predicates identifying locations, and to define an appropriate predecessor relation (to obtain all of the states that can reach a current state by a time step and a discrete transition).

Another development introduced in this paper is the addition of *invariant conditions* to timed automata, which are clock constraints labelling locations of a timed automaton. Control can only remain in the current location while the values of the clocks satisfy the invariant condition, and therefore such conditions can enforce progress between the locations of the model. This type of timed automaton, in which all locations are equipped with invariant conditions, is called *timed safety automata.* As the models subject to the techniques discussed subsequently in this section are timed safety automata, we henceforth call this model simply "timed automata".

Furthermore, Henzinger et al. redefined the timed temporal logic TCTL by replacing the bounded temporal operator syntax of [ACD93] with a *reset quantifier*. Such an operator resets formula clocks, which are referred to exclusively in TCTL formulae, to times at which certain subformulae become true. For example, the property "$\mathsf{response}$ will necessarily be true within 5 time units of $\mathsf{request}$ being true" is represented by the TCTL formula $\forall\Box(\mathsf{request} \rightarrow z.\forall\Diamond(\mathsf{response} \wedge z \leq 5))$, where the formula clock z is reset to 0 whenever $\mathsf{request}$ is true. Finally, [HNSY94] present a symbolic model checking algorithm for timed automata against TCTL properties.

Note that the model checking algorithm of [HNSY94] is based on *backwards* exploration of the state space, as in the untimed symbolic model checking methods of [CES86, BCM$^+$92]. A method for the verification of reachability properties (and invariance properties, their dual) based on *forward* traversal of the state space of a timed automaton, is presented in [DOTY96]. A successor relation that is defined similarly to the predecessor relation of [HNSY94] is iterated successively from the initial state, with each computed state set forming a node of a graph. Then, if this procedure terminates, the resulting finite graph can be checked for the reachability property using standard graph-theoretic techniques. It is also shown how time-bounded reachability properties, which specify that a set of reachability states should be reachable within a certain time bound, can be expressed and verified by this method.

Two significant extensions of this work were later presented. Firstly, Daws and Tripakis [DT98] showed how *abstractions* can be introduced into the verification of timed automata against reachability properties via forward traversal. One method ensures termination of the reachability algorithm, and was based on the idea that certain bounds on state sets referring to clock values above the greatest constant used in the description of the model may be dismissed, a fact which was also used by Alur and Dill to ensure the finiteness of the region graph [AD94]. The other methods involved removing state sets that are contained in others, reducing the number of clocks of the timed automaton (based on ideas from [DY96]), and generating more space-efficient *convex hulls* of state sets.

Secondly, Bouajjani, Tripakis and Yovine presented an on-the-fly algorithm for the verification of properties against a logic of [BLY96] which involves both path quantification and timed automata themselves as operators [BTY97]. As [BLY96] show that this logic subsumes TCTL, the result of [BTY97] also offers a model checking method for TCTL using forward reachability (although some backwards traversal is also necessary). An additional on-the-fly model checking method is presented in [HKV96].

Another proposal addressing the state space explosion inherent in the region graph suggested state space reductions using symbolic methods, which involves a method of solving clock constraints according to rules given by the reachability property in question, and compositional reasoning, was presented in [LPY95b, LPY95a]. These methods concerned the transformation of both the global system (which comprises of a set of parallel timed automata) and the reachability specification by solving problems for each component at a time.

An approach to the problem of ensuring that a given timed automaton exhibits progress are proposed in [Tri98, Tri99b]. Tripakis considers both time progress, which stipulates that time does not converge, and discrete progress, which requires that the model takes discrete transitions infinitely often. Two methods to the problem of progress are advocated: the first, static method involves the definition of structural conditions on classical timed automata which ensure progress, whereas the second, dynamic method involves checking whether a timed automaton can always progress by using a symbolic forward reachability algorithm.

An alternative approach to the verification of timed automata is that based on the bisimulation minimisation technique discussed in Section 2.2. As the exact amount of time that elapses on certain transitions is not always relevant to the satisfaction of a real-time temporal logic formula, we consider a notion of *time-abstract* bisimulation in which the duration of time transitions is abstracted. Note that region equivalence is a time-abstract bisimulation (see [CGP99] for a proof), although one which may be unnecessarily fine for a given verification attempt. Recall that the bisimulation minimisation technique is based on "partition refinement" obtained by successive iteration of predecessor, intersection and set difference operations on state sets [HM00a]; then the smallest state set that an algorithm for computing the time-abstract bisimlation quotient will have to compute will be a region equivalence class. Algorithms employing various techniques for computing the time-abstract bisimulation quotient have been introduced in [ACD$^+$92a, YL93, TY00]. Then, as time-abstract bisimulation preserves TCTL properties, the resulting equivalence quotient can be used for model checking. We note that partition refinement is used in the model checking techniques of [SS95, STA98], which also employ forward reachability analysis.

2.3.4 Extensions of timed automata

A number of extensions of the basic timed automata model have been proposed in the literature, and only a selected handful are summarised here. The model of *updatable timed automata* [BDFP00a, BDFP00b] allows a more liberal clock resetting mechanism than that included in the original definition of timed automata, which stipulated that clocks can be set to 0 only. The motivation for this extension was to permit more convenient modelling of realistic real-time systems. Bouyer et al. note that resetting clocks to integer values, or to values of other clocks does not increase the expressive power of the model, as a bisimilar timed automaton which resets clocks to 0 can always be found [BDFP00b] (for earlier examples of timed automata with such resets, see [HKPV98] and [Yov98]). Similarly, resetting clocks to the value of other clocks plus a constant does not increase the expressive power of timed automata with rectangular clock constraints only. Indeed, for most classes of *nondeterministic* updates, which involve a clock being reset to a value less than or greater than an integer constant, the value of another clock, or the value of another clock plus an integer constant, the decidability of the emptiness problem depends on whether the clock constraints of the timed automaton are rectangular or triangular [BDFP00a] (resulting in decidability or undecidability, respectively). The results concerning the expressiveness of nondeterministic updates presented in [BDFP00b] are based on the addition of "internal" transitions, which are used to show that classes of timed automata with certain types of such updates are weakly bisimilar to classical timed automata (in some cases, the model must be restricted to having only rectangular clock constraints).

The class of timed automata with *additive* clock constraints, such as $x + y = 1$, where x and y are clocks, is considered in [BD00]. In this paper, Bérard and Dufourd show that the language emptiness problem for this class of timed automata is decidable for the case of two clocks, but undecidable for the case of four or more clocks. The intermediate result concerning three clocks is left as an open problem.

A variant of timed automata for which locations and discrete transitions are associated with *costs* is presented in [BFH$^+$01], together with a region graph based algorithm for finding the minimal cost associated with a path which satisfies a reachability property. This work relates to the algorithm of [NTY00], which computes the minimum *time* (rather than the more general notion of cost) which a timed automaton can take to satisfy a reachability property using symbolic representations of state sets.

2.3.5 Timed automata in practice: tools and case studies

The development of algorithms and techniques for the description and analysis of real-time systems modelled as timed automata has been matched by the construction of associated model checking tools, and their successful use in the verification of realistic systems. We will now briefly discuss the two most successful tools in this field, Uppaal, from the teams at Uppsala and Aalborg, and Kronos, developed at Verimag.

The model checking tool Uppaal [LPY97a] employs a forward traversal verification algorithm to verify reachability and invariance properties. The development of this tool has been guided by the criteria of ease of usage and efficiency. The former criterion was addressed by the use of a graphical interface which, together with a simulator, also allows informal validation to be carried out, as it is possible for the user to select and explore a particular dynamic behaviour of the system [LPY97b]. More pertinent to our interests is the model checking technique of the tool, and the methods in which it attempts to address the second development criterion, efficiency. Because verification in Uppaal is limited to reachability and invariance properties, the tool is fully oriented to making the forward traversal algorithm that it employs as efficient as possible.

Furthermore, several additional methods for increasing the efficiency of model checking have been implemented in Uppaal. These include the re-use of portions of the state space when verifying multiple reachability properties (thereby saving time) and storing only one representative state of graph cycles (thereby saving space) [LPY97b]. Finally, the current version of the tool, called Uppaal2k [PL00], uses the space-efficient CDD data structure [LPWY99], and the clock reduction technique of [DY96]. A description of a parallel implementation of Uppaal2k is given in [BHV00].

Many case studies of verification within Uppaal have been published. The systems that have been verified include an audio/video protocol [HSLL97], a power down protocol [HLS99], a bounded retransmission protocol [DKRT97], a collision avoidance protocol [ABL98], the Philips audio control protocol [BGK$^+$96], a steam generator [KP95, Kri99], a manufacturing plant [LPY97b], a gear-box controller [LPY98], and the IEEE1394 root contention protocol [SS00].

The tool Kronos permits a wider range of properties to be verified than Uppaal, including those specified in TCTL. Kronos implements both the backward traversal algorithm presented in [HNSY94] and the forward traversal algorithms of [DOTY96, BTY97]. It also admits the clock reduction strategy of [DY96] via the tool Optikron [Daw98], and the time-abstract bisimulation minimisation technique of [TY00], using the tool `minim`. Furthermore, the tool has recently been extended to allow integration with the Cadp toolset for discrete systems [FGK$^+$96], which permits the time discretisation technique of [BMT99] to be used. Finally, a graphical user interface for Kronos is in the process of development [HY00]. See [Yov97, BDM$^+$98, Yov00] for surveys of Kronos.

This tool has been successfully used in the verification of realistic systems such as the Philips audio control protocol [DY95], MOS circuits [MY96], the STARI chip [BMT99], and the Fast Reservation Protocol with delayed transmission [TY98]. As a final note, there are examples which show that the performance of Kronos is not as impressive as that of Uppaal [BS00], although this is perhaps not surprising given the greater expressive power of the properties that can be verified in Kronos.

Finally, a new model checking tool for the verification of timed automata, which is based on the algorithm of [STA98], is Pmc. This tools has been used successfully to verify systems such as the IEEE1394 protocol [BST00] and synchronisation protocols [BSTV99]. Experiments based on the verification of Fischer's mutual exclusion protocol yield results that are superior to those of Uppaal [STA98].

2.4 Hybrid automata

Given the success of timed automata as models for real-time systems, researchers in the early 1990s began to explore the possibility of using such a framework for the description and analysis of the wider class of *hybrid systems*. Taking their inspiration from both timed automata and the *phase transition systems* of [MMP92], which were a hybrid system generalisation of the timed transition systems of [HMP92], two groups defined independently the model of *hybrid automata* [ACHH93, NOSY93]. This section will describe the theoretical development of this model, and provide an overview of its practical applicability.

2.4.1 Introduction to hybrid automata

As has already been mentioned, hybrid automata are a generalisation of timed automata and phase transition systems. The latter are defined as time transition systems, except that a set of real-valued, continuous variables is now included, which are subject to differential equations, which may be functions of (discrete or continuous) variables [MMP92]. Phase transition systems also feature the discrete/delay transition dichotomy of timed transition systems, in which the set of transitions of the underlying semantic model are partitioned into two types: that of discrete transitions, which are instantaneous, and in which the discrete variables are changed; and that of delay or continuous transitions, which take some amount of time, and in which only continuous variables are changed (according to the aforementioned differential equations).

We now introduce hybrid automata [ACHH93, NOSY93]. Such models represent both the discrete and continuous components of hybrid systems, together with the interdependencies between these components, in the following way. The discrete component is represented by a graph, the vertices of which are called *locations*, and the edges of which are called *control switches*. The continuous component is represented by a finite set of real-valued variables; if we let n be the number of these variables of a particular hybrid automaton, then the continuous component makes transitions through $\mathbb{R}^n$-space, and we refer to the model as having dimension n. Change in the system is described by three types of conditions:

1. flow conditions, which, for any given control mode, determine the continuous change of the continuous component;
2. jump conditions, which can determine discrete change in both the discrete and continuous components (this change will take the form of a control switch); and
3. invariant conditions, which can force discrete change in the discrete component (and so a control switch must be made).

In addition, initial conditions, which specify the initial states of the discrete and continuous components, are defined. Note that it is standard that the initial and invariant conditions are defined with respect to the set of variables, whereas the flow conditions are defined with respect to the set of variables *and their first derivatives.* Furthermore, the jump conditions are defined with respect to the set of variables before and after discrete change; in this way, a jump condition specifies the values of the variables before the control switch, which *enables* the transitions, and the values of the variables after the switch, which determines the values to which the variables can be set to on completion of the transition. Initial, flow and invariant conditions label locations, whereas jump conditions label control switches. Note that a full description of the system at a point in time given by the current location combined with a valuation for all of the hybrid automaton's real-valued variables.

The underlying semantics of a hybrid automaton is as an infinite-state LTS, for which there may be an infinite number of transitions available in each state. As suggested by [MMP92], these transitions are classified as either discrete or continuous transitions.

Note that the timed automata of the previous section are a restricted class of hybrid automata. The discrete component of the previous model was represented as a finite automaton; the continuous component of both timed and hybrid automata are represented by a finite number of real-valued variables, with the continuous transitions that such a component of a timed automata can make being a restricted class of those that the variables of a hybrid automata can make. More precisely, the differential equations given by the flow conditions of a timed automaton must specify that each variable increases at the same rate as real-time.

Hybrid automata were introduced independently by Alur et al. [ACHH93] and Nicollin et al. [NOSY93]. Both of these papers concentrated primarily on a highly restricted subclass of hybrid automaton, later called *multisingular* [Hen96], in which the rate of increase or decrease of each variable is constant when control remains within a given location, and in which other conditions are described in terms of linear constraints. It is shown in [ACHH93] that the emptiness problem for the subclass of such systems which include only two rates is undecidable, but is decidable if the constraints of the model are rectangular (see Section 2.3) and restricted, deterministic variable resetting is used. Nevertheless, semi-decision algorithms for the analysis of such hybrid automata are given in [ACHH93, NOSY93]; both papers include an extension of the backward traversal model checking technique of [HNSY94], while [ACHH93] adapt the minimisation algorithm of [ACD$^+$92b] to the hybrid context, and [NOSY93] presents a forward reachability algorithm. In all cases, the state sets computed at each iteration of these algorithms are generalisations of the restricted form of polyhedra computed by the techniques of [ACD$^+$92b, HNSY94, DOTY96] for timed automata. The results of [ACHH93, NOSY93] are synthesised in the paper [ACH$^+$95], which also includes approximation techniques for hybrid automaton analysis.

We now deal with two main, and related, branches of research in the field of hybrid automata; firstly concerning issues of *decidability*, and secondly concerning *model checking*.

2.4.2 Decidability of hybrid automata

The majority of the decidability results presented for hybrid automata have concerned subclasses of *linear hybrid automata* [AHH96], for which all conditions are defined by linear constraints, and for which the flow conditions can refer to the first derivatives of variables only. Although a number of papers have presented decidability results for multisingular automata or *stopwatch* automata, for which variables can take values 0 or 1 (for example, [BES93, KPSY99]), a particularly significant paper in this field was that of Henzinger et al. [HKPV98]. This paper considers the class of *rectangular automata*, which are linear hybrid automata involving only rectangular constraints (note that this means that flow conditions take the form of convex, rectangular envelopes), as introduced in [PV94]. Puri and Varaiya present a method for showing the decidability of *initialised* rectangular automata, for which variables are reset on a control switch when the flow condition in the switch's source differs from that of the target, via digitisation. An alternative technique is presented in [HKPV98], which shows that every initialised rectangular automaton can be reduced to an initialised multisingular automaton with the same language, which can then be reduced to a bisimilar initialised stopwatch automaton, which can then be reduced to a bisimilar timed automaton. Broadly speaking, as the reachability and language emptiness problems for timed automata are decidable, the reachability and language emptiness problems for initialised rectangular automata are also decidable. Several undecidability results are also presented in [HKPV98], including those showing that the requirement of initialisation (even for stopwatch automata) is necessary for such decidability, as is the requirement of rectangularity of constraints. We note that these results relate to those of updatable timed automata [BDFP00a, BDFP00b] which sometimes require the use of such rectangular constraints (certain classes of such timed automata can be regarded as a form of rectangular automata in which variables are restricted to increase at the same rate as real-time, but the clock reset conditions are permitted to be as liberal as those in the full class of rectangular automata). Furthermore, the result of [BD00], which considers the case of timed automata of additive clock constraints, also relates to the disparity between the the (un)decidability results of linear hybrid automata and rectangular automata.

Similar results to those of [HKPV98] are shown in [OSY94], although here the motivation is more directly oriented towards model checking. The authors show how a multisingular automaton may be constructed from a rectangular automaton; as the former simulates the latter (a result also shown in [HKPV98]), then every TCTL formula with only universal quantification over paths that is true on the multisingular automaton is also true for the rectangular automaton. Therefore, the authors view the multisingular automaton as an *abstraction* of the original, rectangular automaton, that may be used to show positive model checking results for the latter. This relates to the work in the non-hybrid setting of [GL94], as explained in Section 2.2.

The problem of finding finite bisimilarity equivalence relations for hybrid automata is addressed in [Hen95]. In this paper, Henzinger generalises to region equivalence for timed automata to hybrid automata (recall that region equivalence is a time-abstract bisimulation), defining some superclasses of (initialised) multisingular and stopwatch automata for which this is possible. These results are later extended by Lafferiere et al. [LPY99], who show that hybrid automata with certain examples of deterministic flow conditions which cannot be expressed in linear hybrid automata have finite bisimilarity quotients. These results are subject to the condition that all variables are reset on every control switch. The case for the equivalence relation of bi-directional simulation is given in [HHK95], where it is shown that every two-dimensional rectangular automaton has a finite quotient according to this equivalence. The necessity of the restriction to two dimensions for the finiteness of the quotient is shown in [HK96].

2.4.3 Model checking for hybrid automata

One of the earliest model checking strategies for hybrid automata was presented by Henzinger et al. in [AHH96]. This concerned the adaptation of the backwards traversal algorithm of [HNSY94], which is used to verify timed automata against TCTL properties, to the case of verifying linear hybrid automata against ICTL properties. We note that ICTL stands for Integrator Computation Tree Logic, and can be used to express duration requirements through the use of stopwatch variables in reset quantifiers. Although the algorithm presented by [AHH96] is not guaranteed to terminate, the authors note that it will do so for many interesting instances of systems (as borne out by their experiments with the tool HyTech, which will be covered in the next section). Henzinger and Majumdar have recently presented work which shows that, for initialised rectangular automata, this technique is decidable for LTL properties [HM00b] (naturally, these decidability results are based on those of [HKPV98]). Similarly, the results concerning finite bisimulation and simulation quotients, in addition to the language equivalence of initialised rectangular automata and timed automata, are presented in a model checking context in [AHLP00]. A further class of linear hybrid automata for which TCTL model checking is decidable is given in [RR96], and [HR98] offers a technique in which deductive methods can be used in combination with forward traversal to compute the reachable state space of subclasses of linear hybrid automata.

The remainder of this section is devoted to *approximate* methods for the verification of hybrid automata. Henzinger et al. [HHWT98] present two methods for the *conservative approximation* of complex, non-linear continuous behaviour of hybrid systems. The first is based on the replacement of certain non-linear variables with clocks, whereas the second is based on over-approximation of non-linear flow conditions by the polyhedra used in the flow conditions of a linear hybrid automaton.

A method for the translation of linear hybrid automata to (uninitialised) stopwatch automata is presented in [CL00]. This translation is based on the use of multiple locations, control switches, and further continuous variables, to "simulate" the behaviour of the linear hybrid automaton for any given location or control switch. The translation preserves languages (although this is a weak preservation result, as the stopwatch automaton includes internal transitions not included in the original linear hybrid automaton), and therefore preserves reachability properties. The authors then show that the timed automaton model checking tool Uppaal may be used to *over-approximate* the reachable state space of a stopwatch automaton. As Uppaal is a relatively efficient model checking tool, then it is possible that this method may be used for the efficient verification of realistic systems modelled as linear hybrid automata; however, the addition of additional variables, locations and control switches during the translation process is a disadvantage.

2.4.4 Tools for model checking hybrid automata

The most significant tool for the verification of hybrid automata is that of HyTech [HHW97]. This tool is capable of analysing linear hybrid automata, and uses algorithms based on backward traversal [AHH96] and forward traversal [Ho95]. The two inputs of HyTech are a text-based description of the hybrid automaton components of the system to be verified, and a set of analysis commands. These commands take the form of a simple programming language with iteration, which can refer to state sets used in the analysis of the model. Furthermore, operations to find the successor set, or predecessor set, of a particular state set are offered, as are boolean operations. Algorithms for the verification of TCTL, ICTL, LTL, reachability and invariance properties can be coded into this language, although built-in procedures for reachability, the generation of diagnostics, and parametric analysis (based on [AHV93]) are included.

The tool HyTech has been applied, with varying degrees of success, to a number of case studies. Early examples of hybrid systems verified by the authors HyTech are the gas burner, Fischer's mutual exclusion protocol (with drifting clocks), a robot control system [HHW95], the Philips audio control protocol [HW95], and a steam boiler control system [HW96]. Subsequent verification attempts were made by other authors: for instance the aerospace application of [NS95], the collaborative effort by researchers from the Munich University of Technology and Bmw on verifying an automotive control system [SMF97], an engine controller [VWB$^+$98], and a coordination strategy for a team of autonomous robots [AEK$^+$99].

Recently, a prototype of a new tool for hybrid automaton analysis, called HyperTech, has been developed [HHMW00]. This work addresses perceived shortcomings in the HyTech, and proposes the use of interval numerical methods both for the removal of these inadequacies concerning the accumulation of arithmetic overflow errors (which were highlighted particularly in the case study [SMF97]), and the expansion of the scope of the tool to models with more general continuous dynamics. In addition to the linear constraints used in the previous version of HyTech, these continuous dynamics can now be expressed in terms of a combination of polynomials, exponentials and trigonometric functions, with verification being performed using conservative approximation methods. In contrast, initial, invariant and jump conditions are restricted in order to guarantee that the approximation methods are conservative. The tool HyperTech has already been applied to a thermostat, a chemical system, and an air-traffic control protocol in [HHMW00].

2.5 Probabilistic systems

We now turn our attention to formal description techniques and model checking methods for *probabilistic systems*; that is, those systems for which a notion of likelihood or probability is associated with some or all of their transitions. As with early applications of temporal logic to computer science, such as that proposed by Pnueli [Pnu77], initial work on the verification of probabilistic systems concerned deductive, proof based methods. Here, such methods were applied to models comprising both of purely probabilistic choice, or those including both nondeterministic and probabilistic choice (henceforth referred to as *probabilistic-nondeterministic systems*). In particular, we note the approach of Pnueli and Zuck [PZ86, PZ93] used the concept of fairness to model probability in this context. These methods will not be referred to in depth here, and the reader is advised to consult [PZ93, Seg95] for overviews of this field.

The remainder of this section will focus on model checking methods for probabilistic and probabilistic-nondeterministic systems. As the notion of real-time is of vital importance to this thesis, we deal with untimed and timed probabilistic systems in separate sections. We note that probabilistic systems are essentially *Markov chains*, and that probabilistic-nondeterministic systems are equivalent to *Markov decision processes*.

2.5.1 Untimed probabilistic systems

Model checking

A pioneering method for the model checking of probabilistic-nondeterministic systems against LTL specifications, using an automata-theoretic paradigm, was presented by Vardi [Var85]. Here, the notion of property satisfaction was *qualitative*; that is, probabilistic systems were viewed to satisfy an LTL property if the probability measure over its paths that satisfy the property is 1. In terms of the terminology of Vardi, a probabilistic system satisfies the property if *almost all* of its computations satisfy the property. Model checking proceeds by a graph-theoretic analysis of the structure of the probabilistic system.

In order to obtain a solution to the model checking problem for probabilistic-nondeterministic systems, Vardi introduced the key notion of an *adversary* (or *scheduler*, as they were originally called). An adversary is defined to resolve the nondeterministic choices of the system, and is therefore associated with a purely probabilistic structure, on which the model checking problem described in the previous paragraph can be performed. Then the system is said to satisfy an LTL formula if all of its adversaries (more precisely, the purely probabilistic structures induced by those adversaries) satisfy the formula. Vardi also equips probabilistic-nondeterministic systems with a notion of fairness of nondeterministic choice, and so an adversary can be classified as fair; model checking can then be performed over fair adversaries only. It is noted that this model checking method is expensive, as it is triply exponential in the size of the formula.

The complexity of this work was subsequently addressed by Courcoubetis and Yannakakis [CY88, CY95]. A new model checking method for probabilistic systems was introduced, which was based on step-by-step transformation of both the system and the LTL formula, while preserving the probability of the satisfaction of the formula at each step. For probabilistic-nondeterministic systems, an automata-theoretic approach was presented which improved on the method of Vardi [Var85] by a single exponential in the size of the formula.

In a subsequent paper [CY98], Courcoubetis and Yannakakis studied the problem of verifying probabilistic-nondeterministic systems against ω-regular properties (which include properties expressed as automata or LTL formulae). Significantly, the authors presented a method for *quantitative* model checking of such properties; that is, a method for determining whether all adversaries of the model satisfy the property with probability above some threshold is given.

A temporal logic for the specification of quantitative properties, which refer to a bound of the probability of the satisfaction of their subformulae, is given by Hansson and Jonsson in [HJ94]. The authors introduce the logic PCTL, which is interpreted over probabilistic systems, and in which the standard path quantification operators $\exists$ and $\forall$ of CTL are replaced by *probabilistic quantifiers*. For example, if φ is a formula interpreted over paths of the system, and $\lambda \in [0,1]$, then $[\varphi]_{\geq\lambda}$ is true in a state if the probability measure of the set of paths satisfying φ and leaving that state is greater than or equal to λ. The case for $[\varphi]_{>\lambda}$ follows similarly, and properties involving upper bounds on probabilities can be defined using negations of these lower bounded operators. A model checking algorithm is also presented, based on the labelling algorithm of [CES86], with the computation of probabilities necessary for labelling states with formulae involving probabilistic quantification being performed by reduction to a linear programming problem; more precisely, a set of linear equations is solved. This logic is extended to that of PCTL*, which differs from PCTL in the same way that CTL* differs from CTL, in [ASB$^+$95]. A model checking technique for probabilistic systems, based on the method of [CY88], is also sketched in [ASB$^+$95] (more detail is given in [CT00]). Furthermore, Aziz et al. show that model checking for "generalised Markov processes", for which probabilities are not completely specified, is decidable.

A model checking method for probabilistic-nondeterministic systems against PCTL and PCTL* properties is given by Bianco and de Alfaro [BdA95]. Again, the method is based on a state-labelling paradigm. However, when resolving a PCTL formula with probabilistic quantification in a state, rather than calculate the exact probability of the paths leaving the state as in [HJ94], we now identify the *maximal* and *minimal* such probabilities. This is a natural consequence of the addition of nondeterministic choice in this model; the maximal probability corresponds to the adversary which assigns the maximal probability to the satisfaction of the path property, and is used in the verification of formulae which involve *upper* bounds on the probability of satisfaction. Minimal probability having an analogous definition, and are used to verify formulae involving *lower* probability bounds. These probabilities are computed by solving an optimisation problem over system of linear inequalities, rather than linear equations as in [HJ94]. There exists such an inequality for each nondeterministic choice available in a state, for all states; each inequality reflects the probabilistic transition relation of the system. Then, to compute the maximal probability of satisfying a property, a minimisation problem is solved over all such inequalities; conversely, to obtain the minimal such probability, a maximisation problem is solved. The algorithm for the verification of PCTL* presented in [BdA95] is obtained by a reduction to the PCTL model checking problem using a transformation of both the formula and the probabilistic-nondeterministic system in question. Model checking of PCTL formulae is shown to be polynomial in the size of the system and linear in the size of the formula, while PCTL* verification is polynomial in the size of the system and doubly exponential in the size of the formula. An alternative, automata-theoretic method for the verification of PCTL* properties is given by de Alfaro in [dA97b, dA97a], which involves the construction of a deterministic Rabin automaton to represent the linear-time formula inside the scope of a probabilistic quantifier. The complexity of the verification is unaffected by this new strategy. A similar algorithm is also presented in [IN96].

A method for the quantitative verification of probabilistic-nondeterministic systems under the assumption of *fairness* is given by Baier and Kwiatkowska [BK98]. Although their definition of fairness in such systems differs from that presented in [Var85], the authors show how their definition can accommodate the notion presented in this earlier paper. A different, but equivalent syntax for PCTL and PCTL* is given in [BK98], which makes quantification over adversaries explicit, with the resulting logics being called PBTL and PBTL*.

The practical use of such algorithms has primarily been addressed by the definition of data structures for the *symbolic* representation of state sets and transitions. In particular, the extension of BDDs called Multi-Terminal Binary Decision Diagrams (MTBDDs), in which leaves comprising of values other than just 0 and 1 can be included, have formed the basis of the work of Baier et al. [BCH$^+$97], who apply this data structure to probabilistic systems, and define an associated model checking algorithm. This method is extended to probabilistic-nondeterministic systems in [Bai98], and implemented in the tool Prism [dAKN$^+$00]. The paper introducing this tool also reports on a number of methods which can be used to improve the performance of probabilistic model checking, such as precomputing the set of state which satisfy the property with probability 1 (as inspired by [CY95, CY98, dA99b]), and the space-efficient Kronecker representation. A number of randomised algorithms have been successfully verified by Prism [dAKN$^+$00].

Probabilistic preorders and equivalences

Although not directly relevant to this thesis, the fields of deductive verification for probabilistic-nondeterministic systems and probabilistic process algebra have prompted a number of interesting theories that are used in later chapters; in particular, the extension of behavioural preorders and equivalences to the probabilistic context is of importance. Firstly, Larsen and Skou [LS91] present a notion of *probabilistic bisimulation*, which extends the standard notion of bisimulation to a subclass of probabilistic-nondeterministic systems (the "reactive systems" of the classification of [vGSS95]). This equivalence is based on the idea that the relation can be lifted to probability distributions; that is, two distributions are equivalent if they assign the same total probability to the set of states in the same probabilistic bisimulation equivalence class. Then two states are probabilistically bisimilar if every action and distribution that can be chosen by one state can be matched with the same action and an equivalent distribution by the other state, and vice versa. Aziz et al. [ASB$^+$95] show that probabilistic bisimulation preserves PCTL* properties on probabilistic systems.

Jonsson and Larsen [JL91] extend this work by introducing a notion of refinement for probabilistic systems which serves as an adequate notion of *probabilistic simulation* for such systems. In this case two distributions are regarded as being equivalent if they are related by a function which relates probabilistically similar states. Segala and Lynch [SL95] term such functions *weight functions*, and extend probabilistic bisimulation and simulation to the full class of probabilistic-nondeterministic systems. Furthermore, this paper also shows that formulae of an event-based variant of PCTL* are preserved by probabilistic bisimulation on this class of systems. Finally, Segala and Lynch show that, if a probabilistic-nondeterministic system $\mathcal{S}_1$ satisfies a formula of this logic involving only universal quantification over adversaries, and $\mathcal{S}_1$ probabilistically simulates another probabilistic-nondeterministic system $\mathcal{S}_2$, then $\mathcal{S}_2$ also satisfies the property. These results clearly have close connections with those of

[BCG88, GL94] for non-probabilistic systems. Finally, Baier et al. [Bai96, BEM00] present decision procedures for computing the probabilistic bisimulation and simulation classes of probabilistic-nondeterministic systems.

2.5.2 Timed probabilistic systems

We next turn our attention to the formalisms for the description of *timed* probabilistic and probabilistic-nondeterministic systems, and their associated model checking techniques. The methods presented in the remainder of this thesis refer to models operating according to a notion of *dense* time, and therefore we mainly consider such models here. However, we first note the work of [HJ94] is defined according to discrete-time, in which a transition of a probabilistic system is interpreted as taking one time unit in duration. Furthermore, the model checking method presented by Hansson and Jonsson is defined with respect to a "bounded until" operator, which expresses properties of the form "Φ_1 is true until Φ_2 becomes true, and the latter occurs before time t". Such properties are important for the specification of deadline properties; in the context of a probabilistic temporal logic such as PCTL, this means that we have a facility for specifying *soft* deadline properties, for which we can tolerate failure to meet the deadline with some probability. Also of note is the model of probabilistic timed automata of Segala [Seg95], which is a dense time model, but is used in the context of manual verification techniques.

A particularly notable contribution to the area of the verification of probabilistic systems operating in dense time was offered by Alur, Courcoubetis and Dill [ACD91a, ACD91b], who provided a model checking technique for a variant of Generalised Semi-Markov Processes against timed properties. As with timed automata, the system model consists of a finite graph consisting of locations and edges, and a finite set of clocks. Here the clocks are interpreted as decreasing at the same rate as real-time; when at least one clock reaches zero, a discrete transition across an edge to another location is triggered. Such an edge traversal also can have the effect of resetting some or all of the clocks within integer bounded intervals according to continuous probability distributions. The model checking technique proposed by the authors is based on the region graph construction for the analysis of timed automata [AD94, ACD93]. However, the region graph now takes the form of a purely probabilistic system, as opposed to a nondeterministic LTS. This system is then verified against qualitative TCTL properties [ACD91a]; therefore, the interpretation of formula satisfaction proposed by Vardi [Var85], in which a formula is satisfied if *almost all* paths satisfy the property, is used here. In order to model probabilistic concerns, graph-theoretic analysis in combination with fairness requirements are used.

This work is extended to linear-time properties in [ACD91b], which are expressed as deterministic timed automata [AD94]. Furthermore, the system model is extended to admit exponentially distributed and fixed-delay events

The problem of modelling such systems in a *compositional* framework, in which parallel system components can be described in a modular fashion, lead to the introduction of the *stochastic automata* of D'Argenio et al. [DKB98]. Although no method for the quantitative model checking of stochastic automata has been presented, a technique for the translation of stochastic automata to timed automata with deadlines is given in [D'A00a]; naturally, this process removes the probabilistic information from the model, and can therefore be used to verify functional requirements, as opposed to those pertaining to performance issues.

More relevant to our interests is the model of *stochastic transition systems* of de Alfaro [dA97a, dA98b]. Although this framework permits only exponential distributions, either with a specified or unspecified rate, it permits a number of model checking techniques which can be used to verify interesting properties referring to both functional and performance requirements. Note that distributions with an unspecified rate introduce an important feature to the model; that is, the model may be *underspecified* if the rate of a particular transition is not known to the system designer. Naturally, this introduces some nondeterministic choice to the model. The semantics of stochastic transition systems are given as finite-state *timed probabilistic systems* [dA97a], which are probabilistic-nondeterministic systems for which an expected time is associated with each event (where the occurrence of an event is interpreted as corresponding to the choice of a probability distribution over next states). Timed probabilistic systems (and, by extension, stochastic transition systems) can be verified against PCTL* properties [dA97a], and long-run average properties [dA98a]; furthermore, the minimum and maximum reachability time problems [dA99b] can also be solved in this framework. Some of these techniques were first presented in a discrete-time setting in [dA97b].

One drawback of this approach is that deadline properties (whether soft or hard) cannot be verified using timed probabilistic systems, which is a consequence of the model's use of expected time [dA97a, Section 3.2.3]. This problem is addressed by Baier et al. [BKH99], who present a model checking algorithm for continuous-time Markov chains (a purely probabilistic subclass of stochastic transition systems) against properties of a dense-time, probabilistic temporal logic. This work is inspired by the results of Aziz et al. [ASSB96], who showed that such verification is decidable, but did not provide an algorithm with which it could be performed. Baier et al. extend the logic of the earlier paper with a *steady-state* operator which can express long-run properties of the system in question to define the logic CSL. Although verification of the steady-state and unbounded temporal formulae of CSL can be performed in an exact manner, the time-bounded temporal formulae require approximate reasoning. The methods presented in [BKH99] have been improved upon in [BHHK00], using techniques from performance evaluation and bisimulation minimisation (the latter in the style of [BCG88]). These techniques have been implemented in the tool $\mathsf{E} \vdash \mathsf{MC}^2$ (the Erlangen-Twente Markov Chain Checker), which has been successfully applied to the verification of a queueing system and a cyclic server polling system [HKMS00].

Finally, Gregersen and Jensen introduced a model for probabilistic real-time systems which is based on the addition of discrete probability distributions to timed automata [GJ95]. In particular, probability distributions are defined over the edges of a timed automaton, in much the same manner as in the probabilistic timed automata introduced in Chapter 4 of this thesis. The model of Gregersen and Jensen differs from probabilistic timed automata in that edges are labelled with *weights*, rather than probabilities, and *actions*. This allows an enabling condition to be associated with each edge, rather than with a probability distribution over edges. The intuition underlying the behaviour of this model is the following: when control of the system remains in a particular discrete state, there is a choice of letting time elapse or performing an action; then, if an action is chosen, the weights for the *enabled* edges corresponding to that action are used to derive a discrete probability distribution over these edges. Gregersen and Jensen also defined three logics for the specification of properties of this model; the first two are probabilistic variants of Hennessy-Milner logic extended with timing information and probability bounds, whereas the third is based on the probabilistic, timed logic of Hansson [Han94]. Manual proof methods for verifying these logics against the model are presented, although the development of a model checking algorithm is left as an open problem.

Chapter 3

Preliminaries

3.1 Numbers, sets, relations and sequences

The sets of natural and integer numbers are written as $\mathbb{N}$ and $\mathbb{Z}$ respectively. The set of real numbers is denoted by $\mathbb{R}$, and the set of non-negative real numbers is denoted by $\mathbb{R}_{\geq 0}$. We generally use c to denote natural number or integer constants, n and m to denote natural number constants, i, j, k and l to denote variables ranging over the natural numbers, and x, y and z to denote variables ranging over the real numbers. The symbol δ is usually taken to represent time, and as such ranges over the non-negative reals. To denote a probability value (a real value in the interval $[0, 1]$), we typically use the symbol λ. Let AP be a set of atomic propositions; for simplicity, we assume that this set is constant throughout, unless specified otherwise.

Let Q_1 and Q_2 be sets. A *relation* between Q_1 and Q_2 is a subset of $Q_1 \times Q_2$. For $q_1 \in Q_1$, $q_2 \in Q_2$, we generally write $q_1 \mathcal{R} q_2$ instead of $(q_1, q_2) \in \mathcal{R}$. We say that a relation is *on* a set Q if it is a relation between Q and Q. Consider the relation $\mathcal{R}$ on Q: we say that $\mathcal{R}$ is *reflexive* if $q \mathcal{R} q$ for all $q \in Q$; *transitive* if, for all $q_1, q_2, q_3 \in Q$, if $q_1 \mathcal{R} q_2$ and $q_2 \mathcal{R} q_3$ then $q_1 \mathcal{R} q_3$; and *symmetric* if, for all $q_1, q_2 \in Q$, if $q_1 \mathcal{R} q_2$ then $q_2 \mathcal{R} q_1$. A reflexive, transitive relation is called a *preorder*. A reflexive, transitive and symmetric relation $\mathcal{R}$ is called an *equivalence relation*. An equivalence relation is called *finite* with respect to the set Q if it has a finite number of equivalence classes on Q.

Consider a set Q. A *partition* of Q is a set of non-empty, pairwise disjoint subsets of Q, the union of which is Q. A partition is called *finite* if it is a finite set. Observe that there is a one-to-one correspondence between partitions and equivalences of a set Q; that is, a given partition of Q induces a certain equivalence relation on Q, and, conversely, an equivalence relation on Q induces a partition of Q. Let $Q/_{\mathcal{R}}$ be the set of equivalence classes of the equivalence relation $\mathcal{R}$ on Q, and note that $Q/_{\mathcal{R}}$ is a partition of Q. Clearly, the partition $Q/_{\mathcal{R}}$ is finite if and only if $\mathcal{R}$ is finite on Q.

A *$\emptyset$-inclusive partition* of Q is either a partition $\mathcal{C}$ of Q or a union $\mathcal{C} \cup \{\emptyset\}$ where $\mathcal{C}$ is a partition. If Q is a finite set, $\mathcal{C}$ is an $\emptyset$-inclusive partition of Q, and $f : Q \to [0, 1]$

is a function assigning a real value $f(q)$ in the interval $[0,1]$ to each element q of Q, then:

$$\sum_{C \in \mathcal{C}} \sum_{q \in C} f(q) = \sum_{q \in Q} f(q) \ . \tag{3.1}$$

A *sequence* containing elements from a set Q is a finite or infinite ordered list of elements $q_0 q_1 q_2 \cdots$. If the list has a finite number $n \in \mathbb{N}$ of elements, then $q_i \in Q$ for each $0 \leq i \leq n-1$, otherwise $q_i \in Q$ for each $i \in \mathbb{N}$. For a finite sequence $q_0 q_1 q_2 ... q_n$ for some $n \in \mathbb{N}$, a *prefix* of the sequence is the initial fragment $q_0 q_1 q_2 ... q_m$ for some $m \leq n$; similarly, for an infinite sequence $q_0 q_1 q_2 ...$, a prefix of the sequence is the initial fragment $q_0 q_1 q_2 ... q_m$ for some $m \in \mathbb{N}$.

3.2 Models for probabilistic systems

As explained in Chapter 2, formalisms for the modelling of systems exhibiting probabilistic behaviour can be classified as exhibiting either probabilistic choice only, or a mixture of probabilistic and nondeterministic choice. Below, we introduce formalisms for both types of model. Note that the aim of this thesis is to provide a suitable model for probabilistic hybrid systems exhibiting both nondeterministic *and* probabilistic behaviour, and as such our attention will inevitably focus on the latter class.

3.2.1 Markov chains

First, we introduce a Markov chain-based model for systems exhibiting only probabilistic choice. We will refer to such a model simply as a *Markov chain*, and commonly use the adjective "purely probabilistic" to describe it.

Definition 3.2.1 (Markov chain) *A* Markov chain MC *is a pair* $(S, \mathbf{P})$*, where* S *is a set of states, and* $\mathbf{P} : S \times S \to [0,1]$ *is the transition function, which is such that* $\mathbf{P}(s, s') > 0$ *for all states* $s \in S$ *and for at most countably many* $s' \in S$*, and* $\sum_{s' \in S} \mathbf{P}(s, s') = 1$.

A *transition* of MC from state s, denoted by $s \to s'$, consists of a probabilistic choice of a "next-state" s' according to $\mathbf{P}$ such that $\mathbf{P}(s, s') > 0$. Sequences of transitions called *paths* arise by resolving successively the probabilistic choice of "next-state" encoded in the transition function $\mathbf{P}$ of the Markov chain. Formally, a path of MC is a non-empty finite or infinite sequence of transitions:

$$\omega = s_0 \to s_1 \to s_2 \to \cdots \ .$$

We abuse notation by letting the special case $\omega = s$, for some state $s \in S$, also be a path.

The following notation is employed when reasoning about paths. Take any path ω; the first state of ω is denoted by $\mathit{first}(\omega)$, and, if ω is finite, the last state of ω is denoted by $\mathit{last}(\omega)$. The length of a path, $|\omega|$, is defined in the usual way: if ω is the path $\omega = s$, then $|\omega| = 0$; if ω is the finite path $\omega = s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n$, then $|\omega| = n$; if ω is an infinite path, then we let $|\omega| = \infty$. If $k \leq |\omega|$ then $\omega(k)$ denotes the k-th state along ω. If $k < |\omega|$, then let $\omega^{(k)} = s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_k$ ($\omega^{(k)}$ is the k-th prefix of ω). If $\omega = s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n$ is a finite path and $\omega' = s'_0 \rightarrow s'_1 \rightarrow \cdots$ is a finite or infinite path with $\mathit{last}(\omega) = \mathit{first}(\omega')$, then we let the *concatenation* of ω and ω' be the path:

$$\omega\omega' = s_0 \rightarrow s_1 \rightarrow s_2 \cdots \rightarrow s_n \rightarrow s'_1 \rightarrow \cdots$$

Sets of labelled paths are denoted in the following way. $\mathit{Path}_{\mathit{fin}}^{\mathrm{MC}}$ is the set of finite paths of the Markov chain MC, and $\mathit{Path}_{\mathit{fin}}^{\mathrm{MC}}(s)$ is the set of paths ω in $\mathit{Path}_{\mathit{fin}}^{\mathrm{MC}}$ such that $\mathit{first}(\omega) = s$. For any $i \in \mathbb{N}$, we use $\mathit{Path}_{i}^{\mathrm{MC}}$ to denote the paths of length i of MC, and $\mathit{Path}_{i}^{\mathrm{MC}}(s)$ to denote the set of paths ω in $\mathit{Path}_{i}^{\mathrm{MC}}$ such that $\mathit{first}(\omega) = s$. We use $\mathit{Path}_{\mathit{ful}}^{\mathrm{MC}}$ to denote the set of infinite paths of MC. Finally, $\mathit{Path}_{\mathit{ful}}^{\mathrm{MC}}(s)$ is the set of paths ω in $\mathit{Path}_{\mathit{ful}}^{\mathrm{MC}}$ such that $\mathit{first}(\omega) = s$.

For a Markov chain MC, let $\mathcal{F}_{\mathit{Path}^{\mathrm{MC}}}$ be the smallest sigma-algebra on $\mathit{Path}_{\mathit{ful}}^{\mathrm{MC}}$ which contains the sets $\omega\!\uparrow = \{\omega' \in \mathit{Path}_{\mathit{ful}}^{\mathrm{MC}} \,|\, \omega \text{ is a prefix of } \omega'\}$ for all finite paths $\omega \in \mathit{Path}_{\mathit{fin}}^{\mathrm{MC}}$. Sets such as $\omega\!\uparrow$ are variously called "basic cylinders" [Bai98] or "cones" [Seg95] in the literature.

We now define a measure $\mathit{Prob}^{\mathrm{MC}}$ on the sigma-algebra $\mathcal{F}_{\mathit{Path}^{\mathrm{MC}}}$, by first defining the following function on the set of finite paths $\mathit{Path}_{\mathit{fin}}^{\mathrm{MC}}$ of the Markov chain MC.

Definition 3.2.2 *For the Markov chain* MC*, let* $\mathit{Prob}_{\mathit{fin}}^{\mathrm{MC}} : \mathit{Path}_{\mathit{fin}}^{\mathrm{MC}} \rightarrow [0,1]$ *be the mapping inductively defined on the length of paths in* $\mathit{Path}_{\mathit{fin}}^{\mathrm{MC}}$ *as follows. Let* ω *be a path in* $\mathit{Path}_{\mathit{fin}}^{\mathrm{MC}}$.

- *If* $|\omega| = 0$*, then* $\mathit{Prob}_{\mathit{fin}}^{\mathrm{MC}}(\omega) = 1$.
- *If* $|\omega| > 0$*, then, if* $\omega = \omega' \rightarrow s$ *for some* $\omega' \in \mathit{Path}_{\mathit{fin}}^{\mathrm{MC}}$ *and state* $s \in S$*, we let:*

$$\mathit{Prob}_{\mathit{fin}}^{\mathrm{MC}}(\omega) = \mathit{Prob}_{\mathit{fin}}^{\mathrm{MC}}(\omega') \cdot \mathbf{P}(\mathit{last}(\omega'), s)\ .$$

Definition 3.2.3 *For the Markov chain* MC*, the measure* $\mathit{Prob}^{\mathrm{MC}}$ *on* $\mathcal{F}_{\mathit{Path}^{\mathrm{MC}}}$ *is the unique measure such that:*

$$\mathit{Prob}^{\mathrm{MC}}(\omega\!\uparrow) = \mathit{Prob}_{\mathit{fin}}^{\mathrm{MC}}(\omega).$$

It is also possible to show that any union of "basic cylinders" such as $\omega\!\uparrow$ is also measurable by $\mathit{Prob}^{\mathrm{MC}}$ (see [Seg95]). This allows us to present the following lemma concerning Markov chains and probability measures on finite and infinite paths. The lemma is a marginally restricted variant of Lemma 3.1.4 of [Bai98].

Lemma 3.2.4 (cf. [Bai98]) *Let* $\mathrm{MC} = (S, \mathbf{P})$ *be a Markov chain,* $s \in S$ *be a state, and* $\Omega \subseteq \mathit{Path}_{\mathit{fin}}(s)$ *be a set of finite paths such that, for each distinct* $\omega, \omega' \in \Omega$, *we have* $\omega^{(i)} \neq \omega'$ *for any* $0 \leq i < |\omega|$. *Then:*

$$\mathit{Prob}^{\mathrm{MC}}\Big(\bigcup_{\omega \in \Omega} \omega \uparrow\Big) = \sum_{\omega \in \Omega} \mathit{Prob}^{\mathrm{MC}}_{\mathit{fin}}(\omega)\ .$$

When clear from the context, we drop the superscript MC from the notation for paths and probability measures given above.

Finally, we introduce *pointed Markov chains* as Markov chains with a single distinguished initial state.

Definition 3.2.5 (Pointed Markov chain) *A* pointed Markov chain PMC *is a pair* (MC, s) *consisting of a Markov chain* $\mathrm{MC} = (S, \mathbf{P})$ *and a single initial state* $s \in S$.

3.2.2 Concurrent probabilistic systems

Next, we introduce a Markov decision process-based model for systems exhibiting both probabilistic and nondeterministic choice. First, we introduce some notation concerning discrete probability distributions, which, for simplicity, are referred to simply as *distributions.*

Definition 3.2.6 (Distribution) *A* distribution p *over a set* Q *is a function* $p : Q \to [0, 1]$ *such that* $p(q) > 0$ *for at most a countable number of elements* $q \in Q$, *and* $\sum_{q \in Q} p(q) = 1$. *If* p *is a distribution over* Q *and* $p(q) > 0$ *for only one element* $q \in Q$ *(hence* $p(q) = 1$*), then the distribution* p *is called a* Dirac distribution, *and is denoted* $\mathcal{D}(q)$. *We denote the set of distributions over the set* Q *by* $\mu(Q)$.

A distribution p can be extended to a function $p : 2^Q \to [0, 1]$ which assigns to each subset Q' of Q the total probability $p[Q'] \overset{\mathrm{def}}{=} \sum_{q \in Q'} p(q)$. The *support* of a distribution $p \in \mu(Q)$ is denoted by $\mathsf{support}(p)$, and comprises the set of elements $q \in Q$ for which $p(q) > 0$.

We now introduce our model for systems exhibiting both nondeterministic and probabilistic behaviour, called *concurrent probabilistic systems.* This model has its precedents in the probabilistic-nondeterministic systems of [BdA95], the probabilistic automata of [SL95, Seg95], the Markov decision processes of [dA97a], and the identically named model of [BK98, Bai98]. Occasionally, we borrow terms, concepts and algorithms relating to these other models and apply them to concurrent probabilistic systems; instances of this are clearly highlighted in the text, with appropriate citations given.

Definition 3.2.7 (Concurrent probabilistic system) *A* concurrent probabilistic system $\mathcal{S}$ *is a tuple* $(S, \bar{S}, L, \Sigma, \mathit{Steps})$, *where* S *is a (possibly uncountable) set of states,*

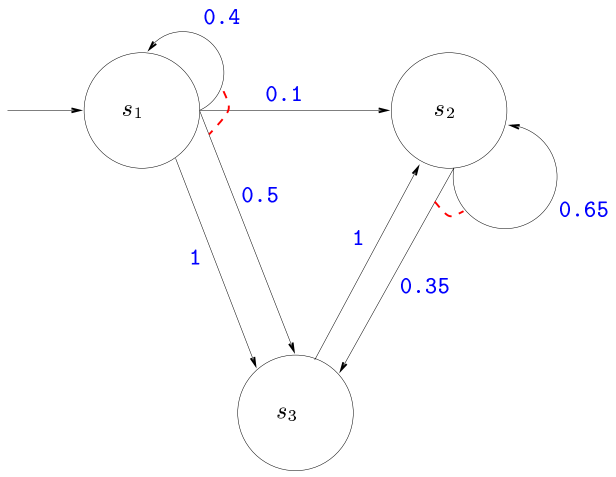


Figure 3.1: A concurrent probabilistic system.

$\bar{S} \subseteq S$ is a set of initial states, $L : S \to 2^{\mathrm{AP}}$ is a function assigning a finite set of atomic propositions to each state, Σ is a set of events, and Steps $: S \to 2^{\Sigma \times \mu(S)}$ is a function which assigns to each state a non-empty set Steps(s) of pairs $(\sigma, p) \in \Sigma \times \mu(S)$ comprising an event σ and a distribution p on S.

We say that a concurrent probabilistic system $\mathcal{S}$ is *finite-state* if its state space S is finite. A concurrent probabilistic system $\mathcal{S}$ has *finite nondeterministic branching* if, for each state $s \in S$, the set *Steps*(s) is finite.

A concurrent probabilistic system $\mathcal{S}$ can be represented diagrammatically as a finite directed graph, provided that it is finite state and has finite nondeterministic branching. The nodes of the graph represent the states of $\mathcal{S}$. An element (σ, p) of *Steps*(s) is represented by a set of outgoing edges from the node representing the state s, where the edges are joined by a dotted arc at their source. Intuitively, such a set of edges represents the distribution p; an individual edge with the target node s' corresponds to a choice of the next state s' according to p. Accordingly, an edge is labelled by a probability value $p(s')$. It is possible to add an additional label to such a set of edges to represent the event σ, but we usually omit this. Furthermore, the initial states of $\mathcal{S}$ are denoted as nodes which are the targets of edges without a source. An example of a concurrent probabilistic system is shown in Figure 3.1. Take the state s_1 (which is also the initial state of the concurrent probabilistic system): then *Steps*$(s_1) = \{(\sigma, p), (\sigma', p')\}$, for some $\sigma, \sigma' \in \Sigma$, and where $p(s_1) = 0.4$, $p(s_2) = 0.1$, $p(s_3) = 0.5$ and $p'(s_3) = 1$.

As with Markov chains, we can define and reason about transitions and paths of a concurrent probabilistic system. A *transition* of $\mathcal{S}$ from state s, denoted by $s \xrightarrow{\sigma,p} s'$, comprises a nondeterministic choice of an event-distribution pair $(\sigma, p) \in$ *Steps*(s), followed by a probabilistic choice of a next-state s' according to p such that

$p(s') > 0$. Paths of a concurrent probabilistic system arise by resolving both the nondeterministic and probabilistic choices. Formally, a *path* of $\mathcal{S}$ is a non-empty finite or infinite sequence of transitions:

$$\omega = s_0 \xrightarrow{\sigma_0, p_0} s_1 \xrightarrow{\sigma_1, p_1} s_2 \xrightarrow{\sigma_2, p_2} \cdots .$$

As in the case of Markov chains, we abuse notation by letting the special case $\omega = s$, for some $s \in S$, also be a path.

For any path ω, the notation *first*(ω), $|\omega|$, $\omega(k)$, and, if ω is finite, *last*(ω), are defined as for Markov chains. The k-th prefix of ω is also defined in a similar manner as in the case of Markov chains: if $k < |\omega|$, then let:

$$\omega^{(k)} = s_0 \xrightarrow{\sigma_0, p_0} s_1 \xrightarrow{\sigma_1, p_1} \cdots \xrightarrow{\sigma_{k-1}, p_{k-1}} s_k .$$

Similarly, if $\omega = s_0 \xrightarrow{\sigma_0, p_0} s_1 \xrightarrow{\sigma_1, p_1} \cdots \xrightarrow{\sigma_{n-1}, p_{n-1}} s_n$ is a finite path and $\omega' = s'_0 \xrightarrow{\sigma'_0, p'_0} s'_1 \xrightarrow{\sigma'_1, p'_1} \cdots$ is a finite or infinite path with *last*(ω) = *first*(ω'), then we let the *concatenation* of ω and ω' be:

$$\omega\omega' = s_0 \xrightarrow{\sigma_0, p_0} s_1 \xrightarrow{\sigma_1, p_1} s_2 \cdots \xrightarrow{\sigma_{n-1}, p_{n-1}} s_n \xrightarrow{\sigma'_0, p'_0} s'_1 \xrightarrow{\sigma'_1, p'_1} \cdots$$

If $k < |\omega|$, then *step*(ω, k) is the event-distribution pair associated with the k-th transition (that is, *step*$(\omega, k) = (\sigma_k, p_k)$). We write $s \xrightarrow{\sigma, p}$ to denote $(\sigma, p) \in$ *Steps*(s) for the state $s \in S$.

For a concurrent probabilistic system $\mathcal{S}$, we use $\mathit{Path}^{\mathcal{S}}_{\mathit{fin}}$ to denote the set of finite paths of $\mathcal{S}$, and $\mathit{Path}^{\mathcal{S}}_{\mathit{fin}}(s)$ to denote the set of paths ω in $\mathit{Path}^{\mathcal{S}}_{\mathit{fin}}$ such that *first*$(\omega) = s$. The meaning of $\mathit{Path}^{\mathcal{S}}_{i}$ and $\mathit{Path}^{\mathcal{S}}_{i}(s)$ for some $i \in \mathbb{N}$ and state $s \in S$, follows naturally from the Markov chain case. The set of infinite paths of $\mathcal{S}$ is written $\mathit{Path}^{\mathcal{S}}_{\mathit{ful}}$, and $\mathit{Path}^{\mathcal{S}}_{\mathit{ful}}(s)$ is the set of paths ω in $\mathit{Path}^{\mathcal{S}}_{\mathit{ful}}$ such that *first*$(\omega) = s$. When clear from the context, we omit the superscript $\mathcal{S}$.

Finally, we introduce *pointed concurrent probabilistic systems* as concurrent probabilistic systems for which the set of initial states is a singleton.

Definition 3.2.8 (Pointed concurrent probabilistic system) *A* pointed concurrent probabilistic system $\mathcal{S} = (S, \bar{S}, L, \Sigma, \mathit{Steps})$ *is a concurrent probabilistic system for which* $|\bar{S}| = 1$.

In order to ease notation, we usually write $(S, \bar{s}, L, \Sigma, \mathit{Steps})$ for the pointed concurrent probabilistic system $(S, \{\bar{s}\}, L, \Sigma, \mathit{Steps})$.

Adversaries of concurrent probabilistic systems

We now introduce *adversaries* of concurrent probabilistic systems as functions which resolve all the nondeterministic choice of the model. We present two classes of adversaries: the class of *deterministic adversaries*, and that of *randomised adversaries*.

Both of these classes have precedents in the relevant literature, such as [Var85, Seg95, dA97a, BK98, Bai98].

Deterministic adversaries. This type of adversary maps every finite path of a concurrent probabilistic system to an event-distribution pair. Deterministic adversaries are inspired by the schedulers of [Var85] and the policies in the Markov decision process literature, but their presentation here has most in common with [BK98, Bai98]. They also agree with the deterministic adversaries of [Seg95] and the deterministic policies of [BdA95, dA97a].

Definition 3.2.9 (Deterministic adversaries) *A* deterministic adversary *of a concurrent probabilistic system $\mathcal{S} = (S, \bar{S}, L, \Sigma, Steps)$ is a function $A : Path_{fin} \to \Sigma \times \mu(S)$ mapping every finite path $\omega \in Path_{fin}$ of $\mathcal{S}$ to an event-distribution pair (σ, p) such that $A(\omega) \in Steps(last(\omega))$. Let Adv be the set of all (deterministic) adversaries of $\mathcal{S}$.*

For a deterministic adversary A of a concurrent probabilistic system $\mathcal{S} = (S, \bar{S}, L, \Sigma, Steps)$, we define $Path^A_{fin}$ to be the set of finite paths such that $step(\omega, i) = A(\omega^{(i)})$ for all $0 \leq i < |\omega|$, and $Path^A_{ful}$ to be the set of paths in $Path_{ful}$ such that $step(\omega, i) = A(\omega^{(i)})$ for all $i \in \mathbb{N}$. Furthermore, $Path^A_{ful}(s)$ is defined to be the subset of paths of $Path^A_{ful}$ such that, for all $\omega \in Path^A_{ful}(s)$, $first(\omega) = s$.

With each deterministic adversary we associate a sequential (in general, infinite-state) Markov chain.

Definition 3.2.10 (Markov chain of a deterministic adversary) *Let A is a deterministic adversary of the concurrent probabilistic system $\mathcal{S}$. Then the* Markov chain of A *is* $\mathrm{MC}^A = (Path^A_{fin}, \mathbf{P}^A)$, *where:*

$$\mathbf{P}^A(\omega, \omega') = \begin{cases} p(s) & \text{if } A(\omega) = (\sigma, p) \text{ and } \omega' = \omega \xrightarrow{\sigma,p} s \\ 0 & \text{otherwise.} \end{cases}$$

Observe that the definition of the set of states of the Markov chain MC^A of a deterministic adversary A as the set of finite paths $Path^A_{fin}$ means that there is a one-to-one correspondence between the paths in this set and the paths of MC^A. More precisely, for every finite path $\omega \in Path^A_{fin}$, we can identify the unique finite path $\pi \in Path^{\mathrm{MC}^A}_{fin}$ such that $\pi = \omega^{(0)} \to \omega^{(1)} \to \ldots \to \omega$. Similarly, for every infinite path $\omega \in Path^A_{ful}$, we can identify the unique infinite path $\pi \in Path^{\mathrm{MC}^A}_{ful}$ such that $\pi = \omega^{(0)} \to \omega^{(1)} \to \ldots$.

Conversely, for every finite path $\pi \in Path^{\mathrm{MC}^A}_{fin}$ such that $|first(\pi)| = 0$ [1], we can identify the unique finite path $\omega \in Path^A_{fin}$ such that $\omega = last(\pi(0)) \xrightarrow{A(\pi(0))} last(\pi(1)) \xrightarrow{A(\pi(1))} \ldots \xrightarrow{A(\pi(n-1))} last(\pi(n))$, where $|\pi| = n$. Finally, for every infinite

[1]Recall that $first(\pi)$ is itself a path in the set $Path^A_{fin}$.

path $\pi \in \mathit{Path}_{\mathit{ful}}^{\mathrm{MC}^A}$ such that $|\mathit{first}(\pi)| = 0$, we can identify the unique finite path $\omega \in \mathit{Path}_{\mathit{ful}}^{A}$ such that $\omega = \mathit{last}(\pi(0)) \xrightarrow{A(\pi(0))} \mathit{last}(\pi(1)) \xrightarrow{A(\pi(1))} \ldots$.

Then, from Definition 3.2.2 and Definition 3.2.3, it follows that, for any deterministic adversary A of $\mathcal{S}$, we can derive a probability measure Prob^A for the Markov chain MC^A. Observe that this is achieved by first defining the auxiliary mapping $\mathit{Prob}_{\mathit{fin}}^{\mathrm{MC}^A}$ on the finite paths of MC^A, and then extending this to the unique measure $\mathit{Prob}^{\mathrm{MC}^A}$. As any finite or infinite path of MC^A is identified with a unique finite or infinite path of the adversary A (that is, a path in the set $\mathit{Path}_{\mathit{fin}}^{A} \cup \mathit{Path}_{\mathit{ful}}^{A}$ of concurrent probabilistic system paths), then $\mathit{Prob}_{\mathit{fin}}^{\mathrm{MC}^A}$ induces the mapping $\mathit{Prob}_{\mathit{fin}}^{A}$ over paths in $\mathit{Path}_{\mathit{fin}}^{A}$, and $\mathit{Prob}^{\mathrm{MC}^A}$ induces the mapping Prob^A over paths in $\mathit{Path}_{\mathit{ful}}^{A}$.

Randomised adversaries. We now introduce randomised adversaries. In contrast to deterministic adversaries, randomised adversaries make a *probabilistic choice over a set of event-distribution pairs* in the last state of a finite path. This class of adversaries agrees with the policies of [BdA95, dA97a], and the randomised adversaries of [Seg95].

Definition 3.2.11 (Randomised adversaries) A randomised adversary *of a concurrent probabilistic system $\mathcal{S} = (S, \bar{S}, L, \Sigma, \mathit{Steps})$ is a function A mapping every finite path $\omega \in \mathit{Path}_{\mathit{fin}}$ of $\mathcal{S}$ to a distribution $p \in \mu(\Sigma \times \mu(S))$ over event-distribution pairs. Let Radv be the set of all randomised adversaries of $\mathcal{S}$.*

The definitions of paths and probability measures of a randomised adversary follow in an intuitive manner. For a randomised adversary A of a concurrent probabilistic system $\mathcal{S} = (S, \bar{S}, L, \Sigma, \mathit{Steps})$, we define $\mathit{Path}_{\mathit{fin}}^{A}$ to be the set of finite paths such that $A(\omega^{(i)})(\mathit{step}(\omega, i)) > 0$ for all $0 \leq i < |\omega|$, and $\mathit{Path}_{\mathit{ful}}^{A}$ to be the set of paths in $\mathit{Path}_{\mathit{ful}}$ such that $A(\omega^{(i)})(\mathit{step}(\omega, i)) > 0$ for all $i \in \mathbb{N}$. Naturally, $\mathit{Path}_{\mathit{ful}}^{A}(s)$ is defined to be the set of paths of $\mathit{Path}_{\mathit{ful}}^{A}$ such that $\mathit{first}(\omega) = s$ for all $\omega \in \mathit{Path}_{\mathit{ful}}^{A}(s)$.

Randomised adversaries can be expressed in terms of infinite-state Markov chains in a similar manner to the class of deterministic adversaries. If A is a randomised adversary of the concurrent probabilistic system $\mathcal{S}$, then $\mathrm{MC}^A = (\mathit{Path}_{\mathit{fin}}^{A}, \mathbf{P}^A)$ is a Markov chain where:

$$\mathbf{P}^A(\omega, \omega') = \begin{cases} A(\omega)(\sigma, p).p(s) & \text{if } \omega' = \omega \xrightarrow{\sigma, p} s \\ 0 & \text{otherwise.} \end{cases}$$

As in the case of deterministic adversaries, it follows from Definition 3.2.2 and Definition 3.2.3 that the function $\mathit{Prob}_{\mathit{fin}}^{A}$ on finite paths and the probability measure Prob^A on finite paths can be defined for any randomised adversary A.

The relationship between deterministic and randomised adversaries. Observe that a deterministic adversary can be stated in terms of a randomised adversary. That is, for a given concurrent probabilistic system $\mathcal{S} = (S, \bar{S}, L, \Sigma, \mathit{Steps})$, a deterministic adversary A is "equivalent" to the randomised adversary A' for which $A'(\omega)(\sigma, p) = 1$ for every finite path $\omega \in \mathit{Path}_{\mathit{fin}}$ of $\mathcal{S}$ and the event-distribution pair $(\sigma, p) \in \mathit{Steps}(\mathit{last}(\omega))$ which is such that $A(\omega) = (\sigma, p)$. Observe that $\mathit{Path}^{A}_{\mathit{fin}} = \mathit{Path}^{A'}_{\mathit{fin}}$, $\mathit{Path}^{A}_{\mathit{ful}} = \mathit{Path}^{A'}_{\mathit{ful}}$, and, for any state $s \in S$, $\mathit{Path}^{A}_{\mathit{fin}}(s) = \mathit{Path}^{A'}_{\mathit{fin}}(s)$ and $\mathit{Path}^{A}_{\mathit{ful}}(s) = \mathit{Path}^{A'}_{\mathit{ful}}(s)$. Furthermore, for every finite path $\omega \in \mathit{Path}_{\mathit{fin}}$ of $\mathcal{S}$, the probability assigned by the measures of the deterministic adversary A and that of the randomised adversary A' to the basic cylinder of ω are equal; that is, $\mathit{Prob}^{A}(\omega\uparrow) = \mathit{Prob}^{A'}(\omega\uparrow)$. Hence, for our purposes, deterministic adversaries and those randomised adversaries which only assign probability 1 to a single event-distribution pair after each finite path can be used interchangeably. This is most convenient when comparing the properties of randomised and deterministic adversaries, which will take place exclusively in Section 9.4 and Appendix A. Elsewhere in the thesis, we will use deterministic adversaries. For simplicity, whenever no confusion will arise, we refer to deterministic adversaries simply as *adversaries*.

3.2.3 State equivalence for probabilistic systems

Next, we introduce two notions of refinement and state equivalence for probabilistic systems, namely those of *probabilistic bisimulation* and *simulation*. As our focus is on systems that exhibit both probabilistic and nondeterministic choice, we recall the standard definitions of the relations for concurrent probabilistic systems, and variants thereof, observing that probabilistic simulation and bisimulation have been defined for purely probabilistic systems in [JL91] and [vGSS95]. In the standard manner, the concept of weight functions [JL91] is used to provide the basis of the definition of simulation, and bisimulation is then defined as a symmetric simulation [SL95]. We write $q \xrightarrow{\sigma,p}$ if there exists a transition $q \xrightarrow{\sigma,p} q'$ for some $q' \in Q$.

Weight functions

Definition 3.2.12 (Weight function cf. [JL91]) *Let $\mathcal{R} \subseteq Q_1 \times Q_2$ be a relation between the sets Q_1, Q_2, and p_1, p_2 distributions such that $p_1 \in \mu(Q_1)$ and $p_2 \in \mu(Q_2)$. A* weight function *for (p_1, p_2) with respect to $\mathcal{R}$ is a function $w : Q_1 \times Q_2 \to [0, 1]$ such that, for all $q_1 \in Q_1$, $q_2 \in Q_2$:*

1. *if $w(q_1, q_2) > 0$, then $(q_1, q_2) \in \mathcal{R}$, and*
2. *$\sum_{q' \in Q_2} w(q_1, q') = p_1(q_1)$, and $\sum_{q' \in Q_1} w(q', q_2) = p_2(q_2)$.*

If there exists a weight function for the distributions $p_1 \in \mu(Q_1), p_2 \in \mu(Q_2)$ with respect to the relation $\mathcal{R} \subseteq Q_1 \times Q_2$, then we write $p_1 \mathcal{R} p_2$.

Intuitively, a weight function for (p_1, p_2) states how the probability assigned by p_1 to an element q_1 of Q_1 may be distributed among the probability assigned by p_2 to the elements of Q_2 that are related to q_1 via the relation $\mathcal{R}$. For example, if $p_1(q_1) = \frac{1}{3}, p_1(q_1') = \frac{2}{3}$, $p_2(q_2) = \frac{1}{2}, p_2(q_2') = \frac{1}{2}$, and $q_1, q_1'\mathcal{R}q_2$, $q_1'\mathcal{R}q_2'$, then a weight function w for (p_1, p_2) with respect to $\mathcal{R}$ is a function such that $w(q_1, q_2) = \frac{1}{3}, w(q_1', q_2) = \frac{1}{6}, w(q_1', q_2') = \frac{1}{2}$. To see this, observe that

$$\begin{aligned}
\sum_{q' \in Q_2} w(q_1, q') &= \frac{1}{3} = p_1(q_1) \ , \\
\sum_{q' \in Q_2} w(q_1', q') &= \frac{1}{6} + \frac{1}{2} = \frac{2}{3} = p_1(q_1') \ , \\
\sum_{q' \in Q_1} w(q', q_2) &= \frac{1}{6} + \frac{1}{3} = \frac{1}{2} = p_2(q_2) \ , \\
\sum_{q' \in Q_1} w(q', q_2') &= \frac{1}{2} = p_2(q_2') \ .
\end{aligned}$$

We now state two general lemmas concerning distributions, relations and weight functions.

Lemma 3.2.13 *Let Q_1, Q_2 be two sets, let $Q_1' \subseteq Q_1$ be a subset of Q_1, and let $Q_2' \subseteq Q_2$ be a subset of Q_2. Let $\mathcal{R}$ be a relation between Q_1 and Q_2 such that, for any $q_1 \in Q_1, q_2 \in Q_2$, if $(q_1, q_2) \in \mathcal{R}$, then $q_2 \in Q_2'$ implies $q_1 \in Q_1'$. If $p_1 \in \mu(Q_1), p_2 \in \mu(Q_2)$ are two distributions such that $p_1\mathcal{R}p_2$, then we have:*

$$\sum_{q \in Q_1'} p_1(q) \geq \sum_{q \in Q_2'} p_2(q) \, .$$

Proof. Let w be a weight function for (p_1, p_2) with respect to $\mathcal{R}$. Take any $q' \in Q_2'$. It follows from Definition 3.2.12 and the assumption of the lemma that, for any $q \in Q_1 \setminus Q_1'$, we have $w(q, q') = 0$. Then $\sum_{q \in Q_1} w(q, q') = \sum_{q \in Q_1'} w(q, q')$, and from this fact and Definition 3.2.12:

$$\sum_{q' \in Q_2'} p_2(q') = \sum_{q' \in Q_2'} \sum_{q \in Q_1} w(q, q') = \sum_{q' \in Q_2'} \sum_{q \in Q_1'} w(q, q') \ .$$

Note that it is possible that $w(q, q') > 0$ for some $q \in Q_1', q' \in Q_2 \setminus Q_2'$. This fact gives us the following inequality:

$$\sum_{q' \in Q_2'} \sum_{q \in Q_1'} w(q, q') \leq \sum_{q' \in Q_2} \sum_{q \in Q_1'} w(q, q') \, .$$

The following equalities are obtained from the commutativity of sum and Definition 3.2.12.

$$\sum_{q' \in Q_2} \sum_{q \in Q_1'} w(q, q') = \sum_{q \in Q_1'} \sum_{q' \in Q_2} w(q, q') = \sum_{q \in Q_1'} p_1(q) \, .$$

Hence $\sum_{q \in Q_1'} p_1(q) \geq \sum_{q \in Q_2'} p_2(q)$. $\square$

Lemma 3.2.14 *Let Q_1, Q_2 be two sets, let $Q'_1 \subseteq Q_1$ be a subset of Q_1, and let $Q'_2 \subseteq Q_2$ be a subset of Q_2. Let $\mathcal{R}$ be a relation between Q_1 and Q_2 such that, for any $q_1 \in Q_1, q_2 \in Q_2$, if $(q_1, q_2) \in \mathcal{R}$, then $q_1 \in Q'_1$ if and only if $q_2 \in Q'_2$. If $p_1 \in \mu(Q_1), p_2 \in \mu(Q_2)$ are two distributions such that $p_1 \mathcal{R} p_2$, then we have:*

$$\sum_{q \in Q'_1} p_1(q) = \sum_{q \in Q'_2} p_2(q)\,.$$

Proof. Let w be a weight function for (p_1, p_2) with respect to $\mathcal{R}$. Observe that $w(q, q') > 0$ only if $q \mathcal{R} q'$ for any $q \in Q_1, q' \in Q_2$. Therefore, by Definition 3.2.12 and the commutativity of sum, we have

$$\begin{aligned}\sum_{q \in Q'_1} p_1(q) &= \sum_{q \in Q'_1}\sum_{q' \in Q_2} w(q,q') = \sum_{q \in Q'_1}\sum_{q' \in Q'_2} w(q,q') = \sum_{q' \in Q'_2}\sum_{q \in Q'_1} w(q,q') \\ &= \sum_{q' \in Q'_2}\sum_{q \in Q_1} w(q,q') \\ &= \sum_{q' \in Q'_2} p_2(q')\,. \qquad \square\end{aligned}$$

State equivalence relations

We now introduce state equivalence relations for concurrent probabilistic systems. The definitions of simulation are taken from [JL91, SL95], whereas the definition of bisimulation is adopted from [LS91, SL95]. Each of the relations is defined with respect to the concurrent probabilistic system $\mathcal{S} = (S, \bar{S}, L, \Sigma, \mathit{Steps})$. First, the definition of simulation is presented. Intuitively, a state s simulates another state s' if any event-distribution pair of a transition from s' can be matched by an event-distribution pair chosen in s such that:

1. the same event is chosen in both, and
2. the distributions are related by a weight function with respect to the simulation relation.

We also require that the states s and s' are labelled with the same set of atomic propositions.

Definition 3.2.15 (Simulation cf. [JL91, SL95]) *A* simulation *of the concurrent probabilistic system $\mathcal{S} = (S, \bar{S}, L, \Sigma, \mathit{Steps})$ is a preorder $\mathcal{R} \subseteq S \times S$ such that, for each $(s_1, s_2) \in \mathcal{R}$:*

1. *$L(s_1) = L(s_2)$, and*
2. *if $s_1 \xrightarrow{\sigma, p_1}$, then $s_2 \xrightarrow{\sigma, p_2}$ for some p_2 such that $p_1 \mathcal{R} p_2$.*

We say that s_2 simulates *s_1, denoted $s_1 \preceq s_2$, if and only if there exists a simulation which contains (s_1, s_2).*

Bisimulation is a symmetric simulation [LS91, SL95], and is now introduced. That is, for bisimilar states, each transition from one state must be matched by a transition from the other state such that the distributions involved are related via weight functions with respect to bisimulation.

Definition 3.2.16 (Bisimulation cf. [LS91, SL95]) *A* bisimulation *of the concurrent probabilistic system $\mathcal{S} = (S, \bar{S}, L, \Sigma, Steps)$ is an equivalence relation $\mathcal{R}$ over S such that, for each $(s_1, s_2) \in \mathcal{R}$:*

1. *$L(s_1) = L(s_2)$,*
2. *if $s_1 \xrightarrow{\sigma, p_1}$, then $s_2 \xrightarrow{\sigma, p_2}$ for some p_2 such that $p_1 \mathcal{R} p_2$; and,*
3. *if $s_2 \xrightarrow{\sigma, p_2'}$, then $s_1 \xrightarrow{\sigma, p_1'}$ for some p_1' such that $p_1' \mathcal{R} p_2'$.*

Two states s_1, s_2 are called bisimilar*, denoted by $s_1 \simeq s_2$, if and only if there exists a bisimulation which contains (s_1, s_2).*

Observe that the distributions that are matched in the above manner must assign equal total probability to each bisimulation equivalence class. As in [Bai98], we note that this suggests the following alternative characterisation of bisimulation, which can be proved using arguments similar to those of [JL91]. The characterisation was used to define probabilistic bisimulation for reactive systems (in which each nondeterministic choice available in a state is associated with a unique event) in [LS91].

Proposition 3.2.17 *Let $\mathcal{S} = (S, \bar{S}, L, \Sigma, Steps)$ be a concurrent probabilistic system. Two states $s_1, s_2 \in S$ are bisimilar if and only if there exists an equivalence relation $\mathcal{R}$ on S such that $s_1 \mathcal{R} s_2$, and, for all states $s, s' \in S$ such that $s \mathcal{R} s'$, we have:*

1. *$L(s) = L(s')$, and*
2. *if $s \xrightarrow{\sigma, p}$, then $s' \xrightarrow{\sigma, p'}$ for some p' such that $p[C] = p'[C]$ for all equivalence classes $C \in S/_{\mathcal{R}}$.*

Observe that, for any states $s_1, s_2 \in S$, if $s_1 \simeq s_2$, then $s_1 \preceq s_2$ and $s_2 \preceq s_1$. Thus, bisimulation is at least as distinguishing (or finer) than the simulation relation.

The definitions of each of these three relations can be adapted to reason about the relation *between* two concurrent probabilistic systems. Without loss of generality, we restrict our attention to pointed concurrent probabilistic systems. The following definition of the disjoint union of such systems is taken from [Bai98], although it is standard in the study of relations between non-probabilistic systems [Mil89].

Definition 3.2.18 (Composed concurrent probabilistic system) *For the two pointed concurrent probabilistic systems, $\mathcal{S}_1 = (S_1, s_1, L_1, \Sigma_1, Steps_1)$, $\mathcal{S}_2 = (S_2, s_2, L_2, \Sigma_2, Steps_2)$, we define the* composed concurrent probabilistic system $\mathcal{S}_1 \uplus \mathcal{S}_2 = (S_1 \uplus S_2, s_1 \uplus s_2, L^{comp}, \Sigma_1 \cup \Sigma_2, Steps^{comp})$ *in the following way. Let $S_1 \uplus S_2$ and $s_1 \uplus s_2$ be the disjoint union of the sets of states, and the initial states, respectively, let $L^{comp}(s) = L_1(s)$ and $Steps^{comp}(s) = Steps_1(s)$ if $s \in S_1$, and let $L^{comp}(s) = L_2(s)$ and $Steps^{comp}(s) = Steps_2(s)$ if $s \in S_2$.*

Then, for a relation $\mathcal{R}$ on $S_1 \uplus S_2$ we write $\mathcal{S}_1 \mathcal{R} \mathcal{S}_2$ if and only if $s_1 \mathcal{R} s_2$ in the composed system $\mathcal{S}_1 \uplus \mathcal{S}_2$.

3.3 Probabilistic Branching Time Logic

3.3.1 Introduction

We now present the probabilistic temporal logic PBTL (Probabilistic Branching Time Logic) [BK98], which can be used to specify properties of systems that exhibit both probabilistic and nondeterministic behaviour, such as concurrent probabilistic systems. Intuitively, PBTL augments the temporal logic CTL with the means to specify bounds on probability. To make this more precise, we recall several pertinent characteristics of CTL formulae. The syntax of CTL involves two categories of formulae: *state formulae*, which are interpreted over states of a labelled transition system, and *path formulae*, which are interpreted over a labelled transition system's paths. A state formula can be obtained from a path formula φ by including φ directly within the scope of either the $\forall$ *universal* path quantifier, or the $\exists$ *existential* path quantifier. As paths of a labelled transition system are obtained by repeated resolution of the nondeterminism inherent in the system's transition relation, we can regard $\forall\varphi$ as being true in a state if and only if φ is true regardless of how nondeterminism is resolved, and $\exists\varphi$ being true if and only if there exists a resolution of nondeterminism which results in φ being true.

For concurrent probabilistic systems, the resolution of *both* nondeterminism and probability results in a path. Furthermore, recall from Section 3.2.2 that an adversary is a function which represents a certain resolution of all of the nondeterministic choices of a concurrent probabilistic system. Therefore, the transference to our context of the notion of quantification over nondeterministic choice can be achieved by quantification over *adversaries*, instead of over paths. We then augment both the universal and existential quantifiers with a probability bound, which can either be strict or non-strict. Then, for a path formula φ, the state formula $[\forall\varphi]_{>\lambda}$ is true in a state of a concurrent probabilistic system if and only if, for all adversaries (that is, no matter how nondeterminism is resolved), the probability measure of the set of paths leaving that state satisfying φ is greater than λ. Similarly, $[\exists\varphi]_{>\lambda}$ is true in a state if and only if there exists an adversary such that the probability of its paths leaving that state and satisfying φ is greater than λ. The meanings of the non-strict versions of these operators, $[\forall\varphi]_{\geq\lambda}$ and $[\exists\varphi]_{\geq\lambda}$, follow immediately. The logic PBTL is then obtained by boolean combinations of such operators. It is shown in [Var85] that the set of paths satisfying the path formulae used in PBTL is in the sigma-algebra $\mathcal{F}_{Path^A_{ful}}$, and therefore its probability measure is well-defined.

We observe that the assertion $[\forall\varphi]_{\geq 1}$ is not exactly the same as universal quantification over all paths, because, in the latter case, we would be including paths of measure 0. A similar argument can be used to show that $[\exists\varphi]_{>0}$ is not exactly the same as existential quantification over paths.

3.3.2 Syntax and semantics

We now introduce the syntax of PBTL. The reader familiar with [BK98] will note that we omit the 'next step' $\bigcirc$ and 'bounded until' $\mathcal{U}^{\leq k}$ path operators. Informally, a path satisfies $\bigcirc\Phi$ if the state reached by its first transition satisfies the PBTL formula Φ, and a path satisfies $\Phi_1\mathcal{U}^{\leq k}\Phi_2$ if a state satisfying Φ_2 is reached after no more than k transitions, and Φ_1 continually holds until then. The 'next step' operator is excluded because we intend to use PBTL as the basis for reasoning about properties of probabilistic hybrid systems, the dense time domain of which renders the standard interpretation of $\bigcirc$ meaningless. More precisely, a given continuous transition can be represented both as a single transition, or as a sequence of transitions of arbitrary length, in the concurrent probabilistic system which represents the semantics of a probabilistic hybrid system. Similarly, the 'bounded until' operator is defined with respect to a bound on the number of transitions made, the literal interpretation of which is again meaningless in our context. However, in Chapter 5, we show how timing information can be added to PBTL, in the style of the dense-time logic TCTL [ACD93, HNSY94], which makes the definition of a *'bounded-time'* analogue of the 'bounded until' operator possible in our framework. Note that a dense-time 'bounded-time until' operator is defined in [ASSB96, BKH99] in the context of continuous-time Markov chains.

Definition 3.3.1 (Syntax of PBTL**)** *The syntax of* PBTL *is defined as follows:*

$$\Phi ::= \texttt{true} \quad | \quad a \quad | \quad \Phi \wedge \Phi \quad | \quad \neg\Phi \quad | \quad [\Phi \;\exists\,\mathcal{U}\; \Phi]_{\sqsupseteq\lambda} \quad | \quad [\Phi \;\forall\,\mathcal{U}\; \Phi]_{\sqsupseteq\lambda}$$

where $a \in \mathrm{AP}$ *is an atomic proposition,* $\lambda \in [0,1]$*, and* $\sqsupseteq$ *is either* $\geq$ *or* $>$.

The formulae of PBTL can be abbreviated following the precedent of similar abbreviations for CTL. The PBTL formulae $[\texttt{true}\forall\mathcal{U}\Phi]_{\sqsupseteq\lambda}$ and $[\texttt{true}\exists\mathcal{U}\Phi]_{\sqsupseteq\lambda}$ are abbreviated by $[\forall\Diamond\Phi]_{\sqsupseteq\lambda}$ and $[\exists\Diamond\Phi]_{\sqsupseteq\lambda}$ respectively. We also write $[\forall\Box\Phi]_{\sqsupseteq\lambda}$ for $\neg[\exists\Diamond\neg\Phi]_{\bar{\sqsupseteq}1-\lambda}$, and $[\exists\Box\Phi]_{\sqsupseteq\lambda}$ for $\neg[\forall\Diamond\neg\Phi]_{\bar{\sqsupseteq}1-\lambda}$, where $\bar{\geq} = >$ and $\bar{>} = \geq$.

For examples of PBTL formulae we refer to the safety, mutual exclusion, and quantitative progress properties adapted from [HJ94, Bai98].

Safety Let $\Phi_{\text{safe}} = \neg[\exists\Diamond\mathsf{error}]_{\geq 0.001}$. If *error* is an atomic proposition corresponding to system error states, then Φ_{safe} is true if it not possible for an error to occur with probability 0.001 or greater.

Mutual exclusion Let $\Phi_{\text{mutex}} = [\forall\Box(\neg\mathsf{crit}_1 \vee \neg\mathsf{crit}_2)]_{\geq 1}$. If crit_i is an atomic proposition corresponding to process i being in its critical section, for $i \in \{1, 2\}$, then Φ_{mutex} is true if, with probability 1, both process 1 and process 2 are never in their critical sections at the same time.

Quantitative progress Let $\Phi_{\text{prog}} = [\forall\Box(\mathsf{request} \to [\exists\Diamond\mathsf{response}]_{\geq 0.99})]_{\geq 1}$. If $\mathsf{request}$ is an atomic proposition corresponding to a system receiving a request, and $\mathsf{response}$ is an atomic proposition corresponding to the system granting a response, then Φ_{prog} is true if, whenever there is a request, then with probability 0.99 or greater, an answer is sent.

The satisfaction relation for PBTL with respect to the states of a concurrent probabilistic system now follows. The cases for `true`, $a \in \text{AP}$, $\wedge$ and $\neg$ are standard for temporal logic. The semantics of the probabilistic operators $[\Phi\, \exists\mathcal{U}\, \Phi]_{\sqsupseteq\lambda}$ and $[\Phi\, \forall\mathcal{U}\, \Phi]_{\sqsupseteq\lambda}$ are defined with respect to the path formula $\Phi\, \mathcal{U}\, \Phi$, the interpretation of which is also standard (see [Eme90], for example).

Definition 3.3.2 (Satisfaction Relation for PBTL) *Given a concurrent probabilistic system $\mathcal{S}$ and a set $\mathcal{A} \subseteq Adv$ of adversaries of $\mathcal{S}$, then for any state s of $\mathcal{S}$, and PBTL formula Φ, the satisfaction relation $s \models_{\mathcal{A}} \Phi$ is defined inductively as follows:*

$$\begin{array}{lll}
s \models_{\mathcal{A}} \texttt{true} & \text{for all } s \in S & \\
s \models_{\mathcal{A}} a & \Leftrightarrow & a \in L(s) \\
s \models_{\mathcal{A}} \Phi_1 \wedge \Phi_2 & \Leftrightarrow & s \models_{\mathcal{A}} \Phi_1 \text{ and } s \models_{\mathcal{A}} \Phi_2 \\
s \models_{\mathcal{A}} \neg\Phi & \Leftrightarrow & s \not\models_{\mathcal{A}} \Phi \\
s \models_{\mathcal{A}} [\Phi_1\, \exists\mathcal{U}\, \Phi_2]_{\sqsupseteq\lambda} & \Leftrightarrow & Prob^A(\{\omega \in Path^A_{ful}(s) \mid \omega \models_{\mathcal{A}} \Phi_1\, \mathcal{U}\, \Phi_2\}) \sqsupseteq \lambda \\
 & & \text{ for some } A \in \mathcal{A} \\
s \models_{\mathcal{A}} [\Phi_1\, \forall\mathcal{U}\, \Phi_2]_{\sqsupseteq\lambda} & \Leftrightarrow & Prob^A(\omega \in Path^A_{ful}(s) \mid \omega \models_{\mathcal{A}} \Phi_1\, \mathcal{U}\, \Phi_2\}) \sqsupseteq \lambda \\
 & & \text{ for all } A \in \mathcal{A} \\
\omega \models_{\mathcal{A}} \Phi_1\, \mathcal{U}\, \Phi_2 & \Leftrightarrow & \text{there exists } i \in \mathbb{N}, \text{ such that } \omega(i) \models_{\mathcal{A}} \Phi_2, \\
 & & \text{and } \omega(j) \models_{\mathcal{A}} \Phi_1 \text{ for all } j \in \mathbb{N} \text{ such that } 0 \leq j < i\,.
\end{array}$$

We say that the pointed concurrent probabilistic system $\mathcal{S} = (S, \bar{s}, L, \Sigma, Steps)$ satisfies the PBTL formula Φ, written $\mathcal{S} \models_{\mathcal{A}} \Phi$, if and only if $\bar{s} \models_{\mathcal{A}} \Phi$.

Observe that the set $\mathcal{A}$ of adversaries used in the satisfaction relation need not be the full set of adversaries Adv of the concurrent probabilistic system $\mathcal{S}$. Instead, it may be appropriate to use a subset of *Adv*; indeed, in Chapter 4, we explain how only a subset of adversaries correspond to realisable behaviour of a probabilistic timed automaton.

3.3.3 Probabilistic temporal logic and state equivalence

It is shown in [SL95] that if a state s simulates another state s', and if s satisfies a property of a probabilistic logic with universal ("for all", or $\forall$) nondeterministic quantification, then s' also satisfies the property. A similar result is shown in [DGJP00]. Therefore, we can view a concurrent probabilistic system $\mathcal{S}$ as a conservative *abstraction* of another concurrent probabilistic system $\mathcal{S}'$ if $\mathcal{S}$ simulates $\mathcal{S}'$. That is, every adversary A of $\mathcal{S}'$ can be emulated by at least one adversary of $\mathcal{S}$ by selecting transitions that match those of A according to the definition of simulation. It is shown below that this notion of emulation is enough to show that, if *all* adversaries of $\mathcal{S}$ satisfy a probabilistic temporal logic property, then all adversaries of $\mathcal{S}'$ must also satisfy the property.

We now introduce $\forall$PBTL as the fragment of PBTL involving only universal quantification over adversaries. Subsequently, we show that $\forall$PBTL properties are preserved by simulation.

Definition 3.3.3 (Syntax of $\forall$PBTL) *The syntax of* $\forall$PBTL *is defined as follows:*

$$\Phi ::= \texttt{true} \;\mid\; \texttt{false} \;\mid\; a \;\mid\; \neg a \;\mid\; \Phi \wedge \Phi \;\mid\; \Phi \vee \Phi \;\mid\; [\Phi \;\forall \mathcal{U}\; \Phi]_{\sqsupseteq \lambda} \;\mid\; [\forall \Box \Phi]_{\sqsupseteq \lambda}$$

where $a \in \mathrm{AP}$ is an atomic proposition, $\lambda \in [0,1]$, and $\sqsupseteq$ is either $\geq$ or $>$.

The satisfaction relation for $\forall$PBTL is obtained by extending that of PBTL (Definition 3.3.2) with four additional rules.

Definition 3.3.4 (Satisfaction Relation for $\forall$PBTL) *Given a concurrent probabilistic system $\mathcal{S}$ and a set $\mathcal{A} \subseteq Adv$ of adversaries of $\mathcal{S}$, then for any state s of $\mathcal{S}$, and* $\forall$PBTL *formula Φ, the satisfaction relation $s \models_{\mathcal{A}} \Phi$ is defined inductively according to the rules in Definition 3.3.2 and the following:*

$$\begin{array}{lll}
s \not\models_{\mathcal{A}} \texttt{false} & \text{for all } s \in S & \\
s \models_{\mathcal{A}} \neg a & \Leftrightarrow & a \notin L(s) \\
s \models_{\mathcal{A}} \Phi_1 \vee \Phi_2 & \Leftrightarrow & s \models_{\mathcal{A}} \Phi_1 \text{ or } s \models_{\mathcal{A}} \Phi_2 \\
s \models_{\mathcal{A}} [\forall \Box \Phi_1]_{\sqsupseteq \lambda} & \Leftrightarrow & Prob^A(\{\omega \in Path^A_{ful}(s) \mid \omega \models_{\mathcal{A}} \Box \Phi_1\}) \sqsupseteq \lambda \\
 & & \text{ for all } A \in \mathcal{A} \\
\omega \models_{\mathcal{A}} \Box \Phi_1 & \Leftrightarrow & \omega(i) \models_{\mathcal{A}} \Phi_1 \text{ for all } i \in \mathbb{N}\,.
\end{array}$$

Then the next theorem follows from the results of [SL95], which concern a probabilistic action-labelled, rather than state-labelled logic.

Theorem 3.3.5 *Let $\mathcal{S}$ be a concurrent probabilistic system, let $\mathcal{A} \subseteq Adv$ be a set of adversaries of $\mathcal{S}$, and let Φ be a $\forall$PBTL formula. If $s_1 \preceq s_2$, for $s_1, s_2 \in S$, then $s_2 \models_{\mathcal{A}} \Phi$ implies $s_1 \models_{\mathcal{A}} \Phi$.*

Proof. See Appendix A. □

Next, as conjectured by [Bai98], bisimulation equivalence implies PBTL equivalence. That is, if two states of a concurrent probabilistic system are bisimilar, then they satisfy the same PBTL formulae. This follows from similar results for the fully probabilistic case [ASB$^+$95], and for a probabilistic action-labelled logic [SL95]. The intuition underlying the result of Aziz et al. and Segala and Lynch, in addition to our presentation of their results, is that, for a pair of bisimilar states, each transition from one state can be matched by a transition from another state, and that the distributions featured in such matched transitions must assign the same total probability to each bisimulation equivalence class. Observing that the definition of bisimulation requires that equivalent states satisfy the same sets of atomic propositions gives us the following result. Note that it is shown in [Bai98] that the reverse of the direction of this theorem does not hold.

Theorem 3.3.6 *Let $\mathcal{S}$ be a concurrent probabilistic system, let $\mathcal{A} \subseteq Adv$ be a set of adversaries of $\mathcal{S}$, and let Φ be a PBTL formula. If $s_1 \simeq s_2$, for $s_1, s_2 \in S$, then $s_1 \models_{\mathcal{A}} \Phi$ if and only if $s_2 \models_{\mathcal{A}} \Phi$.*

Proof. See Appendix A. □

Naturally, such results also apply when comparing two concurrent probabilistic systems. That is, if we compose two pointed concurrent probabilistic systems $\mathcal{S}$ and $\mathcal{S}'$ to obtain $\mathcal{S} \uplus \mathcal{S}'$, then:

- if $\mathcal{S} \preceq \mathcal{S}'$, and $\Phi_\forall$ is a $\forall$PBTL formula, then $\mathcal{S}' \models_{\mathcal{A}} \Phi_\forall$ implies $\mathcal{S} \models_{\mathcal{A}} \Phi_\forall$;
- if $\mathcal{S} \simeq \mathcal{S}'$, and Φ is a PBTL formula, then $\mathcal{S} \models_{\mathcal{A}} \Phi$ if and only if $\mathcal{S}' \models_{\mathcal{A}} \Phi$.

Therefore, we can refer to the preservation of probabilistic temporal logic properties between pointed concurrent probabilistic systems.

3.4 Dense state spaces

Throughout this thesis, the continuous aspects of hybrid systems (including the exact timing attributes of timed systems) will be expressed in terms of points, or sets of points, in the *dense* domain of the reals. This section describes the way in which subsets of such dense spaces can be defined using formulae of particular languages.

We fix a finite set of real-valued variables $\mathcal{X} = \{x_1, ..., x_n\}$. Notation concerning $\mathcal{X}$ will be regularly abused, as sometimes it will be convenient to refer to the index of a variable $x_i \in \mathcal{X}$, whereas, at other times, it is sufficient to omit the index subscript of a variable $x \in \mathcal{X}$. We introduce our notation for individual points in real-valued space. A *valuation* of $\mathcal{X}$ is a vector $\mathbf{a}$ of length n in $\mathbb{R}^n$, which, intuitively, assigns a real value $\mathbf{a}_i$ to each variable $x_i \in \mathcal{X}$. (Equivalently, a valuation can be regarded as a function $\mathbf{a} : \mathcal{X} \to \mathbb{R}$; however, we prefer the former interpretation.) A valuation can also be regarded as defining a point in $\mathbb{R}^n$-space. We write $\mathbf{0}$ for the valuation which assigns 0 to each variable $x \in \mathcal{X}$, and $\mathbf{a} + \delta$ to represent the valuation which assigns $\mathbf{a}_i + \delta$ to each variable $x_i \in \mathcal{X}$, for the valuation $\mathbf{a} \in \mathbb{R}^n$ and the (possibly negative) real value $\delta \in \mathbb{R}$.

We now introduce our methods for the description and manipulation of *sets* of valuations in multidimensional real-valued space. The key object of interest is that of a *polyhedron*; more precisely, a subset of the dense state space that can be described by a quantifier-free formula of the theory $(\mathbb{R}, +, \leq, 0, 1)$ of reals with addition, comparison and integer constants. Let ϕ be a quantifier-free formula of this theory, the (free) variables of which are elements of the set $\mathcal{X}$. For a valuation $\mathbf{a} \in \mathbb{R}^n$, we denote by $\phi[\mathbf{a}]$ the boolean value obtained by substituting the real value $\mathbf{a}_i$ for all variables $x_i \in \mathcal{X}$ in ϕ. Let $[\![\phi]\!]$ be the set of valuations such that $[\![\phi]\!] = \{\mathbf{a} \in \mathbb{R}^n \mid \phi[\mathbf{a}] = \mathtt{true}\}$. Note that the statements $\mathbf{a} \in [\![\phi]\!]$ and $\phi[\mathbf{a}] = \mathtt{true}$ are equivalent. Intuitively, $[\![\phi]\!]$ can be regarded as a *semantic* interpretation of the formula ϕ. A set $\eta \subseteq \mathbb{R}^n$ is *definable* if there exists a quantifier-free formula ϕ of $(\mathbb{R}, +, \leq, 0, 1)$ such that η is the set of all valuations which, when substituted for their appropriate free variables in ϕ, result in the evaluation of ϕ to $\mathtt{true}$. More formally, the set η is definable if there exists a formula ϕ such that $\eta = [\![\phi]\!]$. Note that, for a given set η that is definable, there may exist more than one quantifier-free formula ϕ of $(\mathbb{R}, +, \leq, 0, 1)$ such that $\eta = [\![\phi]\!]$.

We define $\mathsf{Poly}(\mathcal{X})$ to be the set of all subsets of $\mathbb{R}^n$ definable by quantifier-free formulae of the theory $(\mathbb{R}, +, \leq, 0, 1)$. Elements of the set $\mathsf{Poly}(\mathcal{X})$ are referred to as *polyhedra*, and are typically denoted by η. Observe that semantic operations such as intersection, union and complementation, denoted by $\eta \cap \eta'$, $\eta \cup \eta'$, and $\bar{\eta}$, respectively, are well-defined on polyhedra, and have corresponding syntactic transformations in terms of the associated formulae. Difference can also be defined as $\eta \setminus \eta' \overset{def}{=} \eta \cap \bar{\eta}'$. From the fact that the theory of the reals with order, addition, and subtraction is decidable and admits quantifier elimination, there exists a procedure to test the emptiness of a polyhedron. Then, the inclusion test $\eta \subseteq \eta'$ is equivalent to the emptiness test $\eta \setminus \eta' = \emptyset$.

We now identify a number of subclasses of polyhedra, each of which is defined in terms of the types of formulae which define elements of the subclasses.

Convex polyhedra. A polyhedron is *convex* if it can be defined by a formula which is a conjunction of formulae (it is disjunction-free and negation-free). We write

$\mathsf{ConvexPoly}(\mathcal{X}) \subseteq \mathsf{Poly}(\mathcal{X})$ to denote the set of all convex polyhedra.

Zones. A convex polyhedron is a *zone* if it can be defined by a formula which is a conjunction of atomic formulae of the form $x - x' \sim c$, for $x, x' \in \mathcal{X} \cup \{x_0\}$, where x_0 is defined to be a "dummy" clock, the value of which is always 0, $c \in \mathbb{Z} \cup \{\infty\}$ and $\sim \in \{<, \leq\}$. That is, a zone is defined by a conjunction of triangular constraints (see Section 2.3.2). A zone is typically denoted by ζ, and the set of zones is written $\mathsf{Zone}(\mathcal{X})$. Observe that $\mathsf{Zone}(\mathcal{X}) \subseteq \mathsf{ConvexPoly}(\mathcal{X}) \subseteq \mathsf{Poly}(\mathcal{X})$.

Observe that more than one conjunction of formulae may correspond to the same zone. Therefore, it is sometimes convenient to consider the atomic formulae used to describe a zone as being in *canonical form*; that is, when the atomic formulae are as 'tight' as possible (the shortest-path matrix algorithm of Floyd-Warshall was advocated for computing such tightened formulae in [Dil89]). In general, we will assume that all zones referred to is the thesis are in canonical form, unless stated explicitly otherwise.

It is convenient to define the largest constant used in the definition of a zone. Given a formula ϕ, let $c_{\max}(\phi)$ be the maximal absolute integer value $|c| \in \mathbb{Z}$ appearing in an atomic formula which appears in a subformula of ϕ (if no such integer exists, because all of the atomic formulae of ϕ compare differences between clocks to ∞, then we let $c_{\max}(\phi) = 0$). Then, for a zone $\zeta \in \mathsf{Zone}(\mathcal{X})$, the canonical formula of which is ϕ_ζ, we let $c_{\max}(\zeta) = c_{\max}(\phi_\zeta)$.

Rectangles. A convex polyhedron is a *rectangle* if it can be defined by a formula which is a conjunction of atomic formulae of the form $x \sim c$, where $x \in \mathcal{X}$ is a variable, $c \in \mathbb{Z}$ is an integer constant and $\sim \in \{<, \leq, =, \geq, >\}$. A rectangle is typically denoted by γ, and the set of rectangles is written $\mathsf{Rect}(\mathcal{X})$. Observe that $\mathsf{Rect}(\mathcal{X}) \subseteq \mathsf{Zone}(\mathcal{X}) \subseteq \mathsf{ConvexPoly}(\mathcal{X}) \subseteq \mathsf{Poly}(\mathcal{X})$.

A rectangle is *closed* if it can be defined by a formula in canonical form, each conjunct of which is of the form $x \sim c$, where x and c have their usual meanings, and $\sim \in \{\leq, =, \geq\}$. A rectangle is *bounded* if it can be defined by a formula in canonical form which, for every variable $x \in \mathcal{X}$, contains either a conjunct of the form $x = c$, or at least one conjunct of the form $x \sim c$, for $\sim \in \{<, \leq\}$, and at least one conjunct of the form $x \sim' c$, for $\sim' \in \{>, \geq\}$, where x and c have their usual meanings.

If a rectangle γ contains a single point, then γ is called a *singleton*. The projection of the rectangle $\gamma \in \mathsf{Rect}(\mathcal{X})$ onto the axis of the variable $x_i \in \mathcal{X}$ is denoted by γ_i.

An additional operation on valuations, which involves the transformation of one valuation to a rectangle by resetting the values of certain variables, is now defined.

Firstly, we define the *reset operation* on valuations with respect to rectangles. Let $X \subseteq \mathcal{X}$ be a subset of variables, $\mathbf{a} \in \mathbb{R}^n$ be a valuation, and $\gamma \in \mathsf{Rect}(\mathcal{X})$ be a rectangle. Then $\mathbf{a}[X := \gamma]$ is the set of valuations (actually a rectangle) such that $\mathbf{a}' \in \mathbf{a}[X := \gamma]$ if and only if, for all $x_i \in X$, we have $\mathbf{a}'_i \in \gamma_i$, and for all other $x_i \in \mathcal{X} \setminus X$, we have $\mathbf{a}'_i = \mathbf{a}_i$. We often use two special cases of the reset operation. Firstly, if $\gamma \in \mathsf{Rect}(\mathcal{X})$ is a singleton (that is $\gamma \in \mathbb{Z}^n$, then $\mathbf{a}[X := \gamma]$ is also a singleton (equivalently, a valuation), and is termed an *integer reset.* For example, if $\gamma \in \mathsf{Rect}(\mathcal{X})$ is a singleton containing the valuation $\mathbf{0}$, then $\mathbf{a}[X := \mathbf{0}]$ denotes the valuation obtained by setting each variable $x_i \in X$ to 0, and for which all other variables $x_i \in \mathcal{X} \setminus X$ retain their original value $\mathbf{a}_i$. Such a valuation will often be written simply as $\mathbf{a}[X := 0]$.

Now, we define the reset operation on polyhedra. For a set of variables $X \subseteq \mathcal{X}$, a polyhedron $\eta \in \mathsf{Poly}(\mathcal{X})$, and a rectangle $\gamma \in \mathsf{Rect}(\mathcal{X})$, we write $\eta[X := \gamma]$ to denote the polyhedron defined by $\eta[X := \gamma] = \{\mathbf{a}[X := \gamma] \in \mathbb{R}^n \mid \mathbf{a} \in \eta\}$. Similarly, we write $[X := \gamma]\eta$ to denote the polyhedron defined by $[X := \gamma]\eta = \{\mathbf{a} \in \mathbb{R}^n \mid \mathbf{a}[X := \gamma] \subseteq \eta\}$.

For zones, we make use of *forward* and *backward diagonal projections.* Define the *forward diagonal projection* of a zone $\zeta \in \mathsf{Zone}(\mathcal{X})$ to be the zone $\nearrow\zeta$ in $\mathsf{Zone}(\mathcal{X})$, such that $\mathbf{a} \in \nearrow\zeta$ if and only if $\exists \delta \in \mathbb{R}_{\geq 0} \,.\, \mathbf{a} - \delta \in \zeta$. Similarly, we define the *backward diagonal projection* of $\zeta \in \mathsf{Zone}(\mathcal{X})$ to be $\swarrow\zeta$ in $\mathsf{Zone}(\mathcal{X})$, such that $\mathbf{a} \in \swarrow\zeta$ if and only if $\exists \delta \in \mathbb{R}_{\geq 0} \,.\, \mathbf{a} + \delta \in \zeta$.

Chapter 4

Probabilistic timed automata

4.1 Introduction

This chapter introduces a modelling formalism for the description and analysis of probabilistic real-time systems. Our approach is to combine established approaches for real-time system description with those for describing systems which exhibit nondeterministic and probabilistic behaviour.

We present the following motivating example. Say we wished to model formally the behaviour of an 'intelligent light switch' [LP00], which has three operating modes: either the light could be off, or on with either moderate or bright light. When the light is off, the user can turn it on by pressing the switch; if it is pressed twice in quick succession, then the light increases in brightness. A subsequent press of the switch, either after a sufficient delay if the light is on with moderate brightness, or at any time when the light is on at maximum brightness, has the effect of turning the light off, returning the system to its original operating mode.

Consider modelling the light switch with a traditional labelled transition system, in which the modes of operation are represented by states, and the changes between these modes are represented by state transitions. Therefore, we construct a labelled transition system with the three states, off, moderate and bright, which correspond to the operating modes of the switch in an obvious manner, and an appropriate set of transitions. For example, the state moderate has a transition to bright, which represents the switch being pressed when the light is on, and the light becoming bright; it also has a transition to off, which represents the light turning off when the switch is pressed. This labelled transition system is shown in Figure 4.1. Observe that the choice in the state moderate between making the light brighter, and turning the light off, is nondeterministic in nature.

Note that the real-time properties of this system are implicit in the labelled transition system model. In particular, the exact quantity of time that may pass before a second press of the switch is interpreted as a signal to turn the light off, rather than to make it brighter, is not represented. Say this quantity is three seconds. Then, the system can be modelled as a *timed automaton* [AD94, HNSY94] in the following way. Firstly, we equip our labelled transition system with a real-valued variable x, which increases at the same rate as real-time, and, as such, is called a *clock*. Secondly, we distinguish between the times at which the transitions leaving the state moderate are enabled; that is, the transition to bright is enabled before or when three seconds have elapsed since the light was switched on, whereas the transition to off is enabled after three seconds have elapsed. We model this by resetting the clock x to 0 when the state moderate is entered, and enabling the transition to bright only when $x \leq 3$, and the transition to off when $x > 3$. The resulting timed automaton is shown in Figure 4.2.

Now consider the case in which we wish to model the fact that the light switch is faulty, but in which we can disregard exact timing information; when pressed, the signal may only be transferred to the light with probability 0.99. The resulting system can be represented by the concurrent probabilistic system in Figure 4.3.

The remaining problem concerns the way in which the information concerning the fault may be combined with that concerning the exact timing of the system. Observe that, in the timed automaton in Figure 4.2, the exact time for the transition from moderate to off, or from moderate to bright, is nondeterministically chosen. As our intuition concerning the system tells us that the exact time at which the switch is pressed does not affect the probability of switch failure, we can retain the idea that the time at which the switch is pressed is a nondeterministic choice, yet, when that choice has been made, then the subsequent choice as to which mode to go to could be made probabilistically. Then, the probabilistic information concerning the occurrence of a fault and the quantitative timing information concerning the pressing of the switch can be combined into a single model in a natural manner, which is shown in figure Figure 4.4. We call such a model a *probabilistic timed automaton*. In contrast to timed automata, in which the transitions of a labelled transition system are guarded by conditions on clocks, probabilistic timed automata augment concurrent probabilistic systems by associating guards with the *distributions* over transitions.

Observe that, in this particular example, the probabilistic aspects of the system are orthogonal to its real-time aspects. That is, the probability of the occurrence of a fault

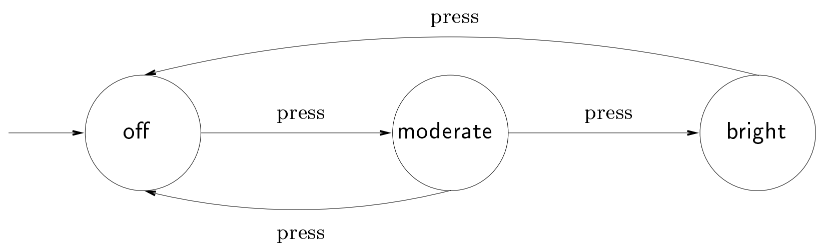


Figure 4.1: A labelled transition system modelling an intelligent light switch.

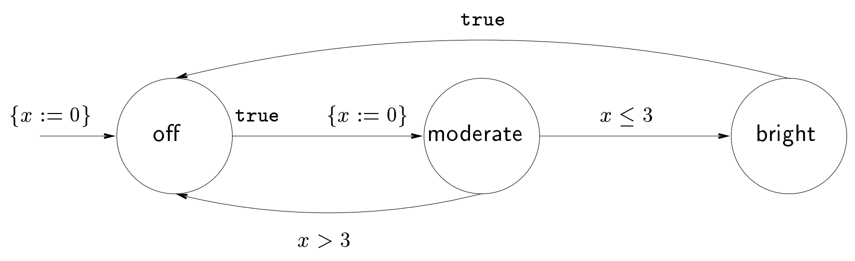


Figure 4.2: A timed automaton modelling an intelligent light switch.

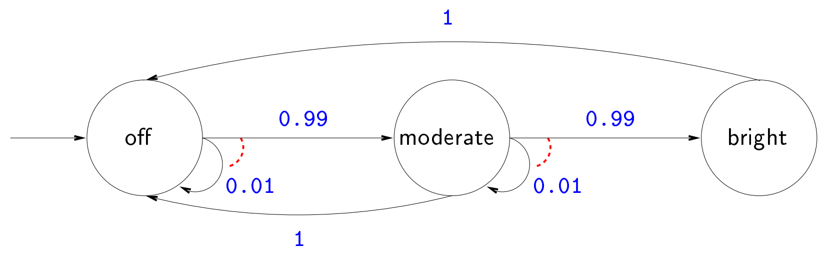


Figure 4.3: A concurrent probabilistic system modelling a faulty intelligent light switch.

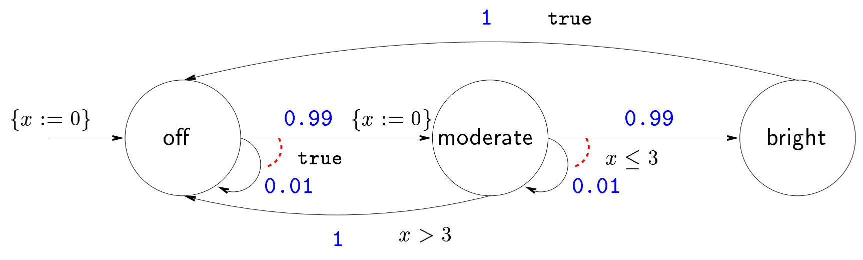


Figure 4.4: A probabilistic timed automaton modelling a faulty intelligent light switch.

is dependent on which mode the system is currently operating in, *not* on the exact timing information. Therefore, the probability of a transition representing a switch failure is dependent only on the location in which the probabilistic timed automaton currently resides, and is independent on either the absolute time given since the start of system execution, or the relative time since a certain discrete transition given by the exact value of the clock x. The orthogonality of time and probability is inherent in probabilistic timed automata; nevertheless, we propose that the above example, and that of Section 4.2.3, while simple, demonstrate the appeal of the model for representing certain probabilistic, real-time phenomena. Furthermore, strategies such as *splitting* of locations could be used to obtain some dependence between the values of clocks and the probabilities relating to certain events.

The contents of this chapter are as follows. Section 4.2 introduces the syntax of probabilistic timed automata, and explains the informal semantics of the model. An example of a probabilistic timed automaton modelling a simple communication protocol is given, and two well-formedness assumptions on the model are discussed. Section 4.3 formally defines the semantics of probabilistic timed automata in terms of infinite-state concurrent probabilistic systems, and Section 4.4 introduces notions of *progress* of the model, which is a way to reason about the realisability of the model's behaviour. Note that substantial parts of this chapter have appeared in [KNSS99, KNSS00a].

4.2 Syntax of probabilistic timed automata

This section introduces formally *probabilistic timed automata* as a modelling framework for real-time systems with probability; this formalism is derived from timed automata and concurrent probabilistic systems. We obtain our model by extending timed automata with discrete probability distributions over discrete transitions (the edges in the graph of a timed automaton), so that the choice of the next discrete state of the automaton (the nodes in a timed automaton's graph) is now both probabilistic and nondeterministic in nature.

4.2.1 Definition of probabilistic timed automata

We proceed to define formally the model of probabilistic timed automata. In the following, we use 2^{Q}_{fn} to denote the finite, non-empty powerset of the set Q.

Definition 4.2.1 (Probabilistic timed automaton) *A* probabilistic timed automaton *$T = (\mathcal{X}, V, \mathcal{L}, init, inv, prob, \langle pre_v \rangle_{v \in V})$ comprises the following components:*

Clocks. *$\mathcal{X} = \{x_1, ..., x_n\}$ is a finite set of* clocks.

Locations. *V is a finite set of* locations.

Labelling function. *$\mathcal{L} : V \to 2^{\mathrm{AP}}$ is a function assigning a set of atomic propositions to each location.*

Initial condition. *$init : V \to \mathbb{N}^n \cup \{\emptyset\}$ is a function that maps every location to an* initial condition*, which is either in $\mathbb{N}^n$ or is the empty set $\emptyset$.*

Invariant set. *$inv : V \to \mathsf{Zone}(\mathcal{X})$ is a function that maps every location to an* invariant set*, which is a zone in $\mathsf{Zone}(\mathcal{X})$.*

Probability distributions. *$prob : V \to 2_{fn}^{(\mu(V \times (\mathbb{N} \cup \{\perp\})^n))}$ is a function that maps every location to a finite, non-empty set of distributions over the set of locations, and a point in $(\mathbb{N} \cup \{\perp\})^n$-space. Therefore, each location v will have a set of associated probability distributions, denoted by $prob(v) = \{p_v^1, ..., p_v^m\}$ for some finite $m \geq 1$.*

Pre-condition sets. *For each $v \in V$, the function $pre_v : prob(v) \to \mathsf{Zone}(\mathcal{X})$ maps every probability distribution associated with a location to a* pre-condition set*, which is represented as a zone in $\mathsf{Zone}(\mathcal{X})$.*

Note that we consider a class of probabilistic timed automata in which the enabling of discrete transitions, and the invariant conditions, may be dependent on both the values of the model's clocks and the *difference* between clock values. That is, we define probabilistic timed automata as having *triangular* clock constraints (see Section 2.3.2).

Note that elements such as $post \in (\mathbb{N} \cup \{\perp\})^n$, where $p(v', post) > 0$, for some distribution $p \in prob(v)$ and locations $v, v' \in V$, are called *post conditions.* The intuition underlying a post condition is that, for a clock $x_i \in X$, if $post_i \in \mathbb{N}$, then x_i is reset to the value $post_i$, otherwise x_i retains its current value. We restrict the values to which clocks can be reset in post conditions to non-negative integer constants. Although more general variants of timed automata exist, for example those for which clocks can be reset to values of other clocks [Yov98], or the class of *updatable timed automata* [BDFP00a, BDFP00b], for which clocks can be reset to values within intervals (whether open or closed, bounded or unbounded), we choose not consider these classes here. We conjecture that the verification results presented can be extended to probabilistic timed automata with the conditions that reset clocks to values of other clocks, and to certain classes of updatable timed automata extended with probability distributions.

Often it is convenient to regard the discrete transitions of a probabilistic timed automaton T as relating to *edges* between its locations. Formally, an edge e is a tuple of the form $(v, v', post, p) \in V \times V \times (\mathbb{N} \cup \{\perp\})^n \times \mu(V \times (\mathbb{N} \cup \{\perp\})^n)$. We define the *set E of edges* of the probabilistic timed automaton T such that $(v, v', post, p) \in E$ if and only if $p \in prob(v)$ and $p(v', post) > 0$ for some $v \in V$. For any $v \in V$, the set $\mathsf{out}(v)$ contains all edges of the form $(v, \_, \_, \_)$, and the set $\mathsf{in}(v)$ contains all edges of the form $(\_, v, \_, \_)$ In addition, for any $v \in V$ and $p \in prob(v)$, the set $\mathsf{edge\_support}(p)$ contains all edges of the form $(\_, \_, \_, p)$. Intuitively, the edges in $\mathsf{edge\_support}(p)$ are those which correspond to the location v and the tuples in $\mathsf{support}(p)$.

We use $\mathsf{Zone}(T) \subseteq \mathsf{Zone}(\mathcal{X})$ to denote the set of all zones appearing as an invariant condition or pre-condition of the probabilistic timed automaton T. Formally, let:

$$\mathsf{Zone}(T) = \{\{pre_v(p) \mid p \in prob(v)\} \cup inv(v) \mid v \in V\} .$$

Now let $Post(T) \subset \mathbb{N}^n$ be the set of post conditions of T. Formally, we let:

$$Post(T) = \{post \in (\{\mathbb{N} \cup \{\bot\})^n \mid (v', post) \in \mathsf{support}(p), p \in prob(v), v \in V\} .$$

Let $c_{\max}(Post(T))$ be the maximal natural number appearing in a post-condition of T; that is, let $c_{\max}(Post(T)) = \max(\{post_i \in \mathbb{N} \mid post \in Post(T), x_i \in \mathcal{X}\})$. Finally, let

$$c_{\max}(T) = \max(\{c_{\max}(\zeta) \mid \zeta \in \mathsf{Zone}(T)\}, c_{\max}(Post(T)))$$

be the maximal constant used in the description of T. In Chapter 5, this constant will be used to ensure the decidability of our model checking methods.

Let $\mathsf{Zone} \in \mathsf{Zone}(\mathcal{X})$, $post \in (\mathbb{N} \cup \{\bot\})^n$, and let $X \subseteq \mathcal{X}$ such that $post_i \in \mathbb{N}$ for each $x_i \in X$. We abuse notation by the using $\zeta[X := post]$ to denote the reset operation $\zeta[X := post']$, where $post'_i = post_i$ for each $x_i \in X$, and $post'_i$ is arbitrary for all other clocks. The reverse operation $[X := post]\zeta$ has an analogous definition.

4.2.2 Behaviour of probabilistic timed automata

We now give an informal description of the semantics of probabilistic timed automata; a formal treatment will be given in Section 4.3. First, recall several pertinent characteristics of the semantics of *non-probabilistic* timed automata. The state of a timed automaton at any point in execution is fully described by the current location and the values of its clocks. The behaviour of this model comprises a sequence of transitions, some of which correspond to movement between locations, and some of which correspond to the location remaining constant, but with the values of the clocks increasing at the same rate as real-time. The set of transitions available in a given state is determined by comparing the current values of clocks with the *enabling* conditions and *invariant* conditions. Each transition can be classified in one of two ways: either as a *time transition*, which involves some non-negative amount of time elapsing, or as a *discrete transition*, which corresponds to an edge of the timed automaton's graph being crossed.

Probabilistic timed automata differ from this in the following respect: the current values of the clocks are compared with enabling and invariant conditions to decide which time transitions and which *probability distributions* are available for nondeterministic choice. Intuitively, we have replaced the notion of edges with the notion of probability distributions *over edges*. The caveat to this observation is that we now associate enabling conditions with probability distributions, rather than with the individual edges in their support; this is simply to prevent the pathological situation in which not all of the edges in the support of a distribution being used to make a discrete transition are enabled. We note that an alternative approach is taken in [GJ95] (see Section 2.5.2).

We return to the explanation of the behaviour of probabilistic timed automata. A *state* of a probabilistic timed automaton $T = (\mathcal{X}, V, \mathcal{L}, \mathit{init}, \mathit{inv}, \mathit{prob}, \langle \mathit{pre}_v \rangle_{v \in V})$ is defined informally as a pair $(v, \mathbf{a})$ consisting of a location $v \in V$, and a valuation $\mathbf{a} \in \mathbb{R}^n_{\geq 0}$ such that $\mathbf{a}$ satisfies the location's invariant condition; that is, we have $\mathbf{a} \in \mathit{inv}(v)$. Then, in any state of the probabilistic timed automaton, the possible transitions take the following form.

- Firstly, a *time transition* of duration $\delta \in \mathbb{R}_{\geq 0}$ is possible if and only if the clock values obtained by adding δ to each of the current values satisfy the invariant condition associated with the current location. More formally, if $(v, \mathbf{a})$ is the current state of the system, then a time transition of duration $\delta \in \mathbb{R}_{\geq 0}$ to the state $(v, \mathbf{a} + \delta)$ is possible if $\mathbf{a} + \delta \in \mathit{inv}(v)$.

- Secondly, a *probabilistic (discrete) transition* corresponding to the distribution p is possible if and only if p belongs to the set of distributions associated with the current location, and the current clock values satisfy the pre-condition of p. That is, if $(v, \mathbf{a})$ is the current state of the system, then a probabilistic transition corresponding to the distribution p can be performed if $p \in \mathit{prob}(v)$ and $\mathbf{a} \in \mathit{pre}_v(p)$. The model then completes the transition by making a probabilistic choice according to p. That is, for any location $v' \in V$, post condition $\mathit{post} \in (\mathbb{N} \cup \{\bot\})^n$, the probability that the probabilistic timed automaton makes a discrete transition to v', while simultaneously resetting each clock x_i to the value post_i if $\mathit{post}_i \in \mathbb{N}$, is given by $p(v', \mathit{post})$.

Observe that, at a point of execution of a probabilistic timed automaton, a nondeterministic choice is made amongst the set of transitions obtained by taking the union of the possible time transitions and probabilistic transitions. If a time transition is chosen, then the target state is unique; however, if a probabilistic transition is chosen, then the target state depends on a probabilistic choice.

4.2.3 Example of a probabilistic timed automaton

We now give a further example of a probabilistic timed automaton, this time modelling a simple communication protocol operating with an unreliable, lossy channel. The diagrammatic representation of the probabilistic timed automaton is shown in Figure 4.5. The system consists of a sender and a receiver which communicate over a single channel, and two global clocks x and y (we make the assumption of the existence of global clocks for simplicity). Transit across the medium is instantaneous. Operation of the protocol commences with the delivery of new data to the sender, and with both clocks x and y set to 0. The sender retains the data for between 2 and 3 time units before transmitting it onto the medium and resetting the clock x to 0. With probability 0.95, the data is correctly received, in which case the receiver waits for not more than 1 time unit to attempt to return an acknowledgement to the sender. This attempt is successful with probability 0.99, after which an arbitrary length of time may elapse before another data packet is delivered to the sender for transmission across the medium. However, if either the acknowledgement or the original data packet was lost by the channel, then the receiver is idle whereas the sender is still waiting for an acknowledgement; therefore, the sender attempts to re-send the data at a frequency varying between 2 and 3 time units. If exactly 7 time units have elapsed since the data first arrived at the sender, or since the receiver last received the data, then the system aborts.

This communication protocol is modelled by the probabilistic timed automaton T_{comm} in the following way. Consider the initial node, $\mathsf{s} : \mathsf{hasData}$, $\mathsf{r} : \mathsf{idle}$, henceforth abbreviated to hi. The invariant of hi is the zone described by the constraint written in the body of the node, $x \leq 3$, which represents the fact that the clock x is not allowed to exceed 3. The node hi has one probability distribution associated with it, as represented by the edges connected by a dashed arc, which is only enabled when $x \geq 2$. Therefore, when x is between 2 and 3, this distribution, which corresponds to the sender transmitting the data onto the medium, may be nondeterministically chosen. Furthermore, if x equals 3 then the invariant condition requires that the distribution *must* be taken. This distribution is defined over two edges: the first leads to $\mathsf{s} : \mathsf{waitAck}$, $\mathsf{r} : \mathsf{received}$ (abbreviated to wr), and corresponds to the successful delivery of the data across the medium; as such, it is labelled with the probability 0.95. The second edge leads to $\mathsf{s} : \mathsf{waitAck}$, $\mathsf{r} : \mathsf{idle}$ (abbreviated to wi), and represents loss of data with probability 0.05. If the former edge is taken, both clocks x and y are reset, whereas, in the latter case, only x is reset, as denoted by the edge labels $\{x, y := 0\}$ and $\{x := 0\}$ respectively. More formally, $p \in \mathit{prob}(\mathsf{hi})$, where $p(\mathsf{wr}, (0, 0)) = 0.95$, $p(\mathsf{wi}, (0, \perp)) = 0.05$. Another aspect of probabilistic timed automaton behaviour can be witnessed in the node wi: it is possible for *both* of the distributions associated with wi, namely $p'_1, p'_2 \in \mathit{prob}(\mathsf{wi})$, to be enabled at the same time. In such a case, there can be a nondeterministic choice between choosing p'_1 and choosing p'_2. From this

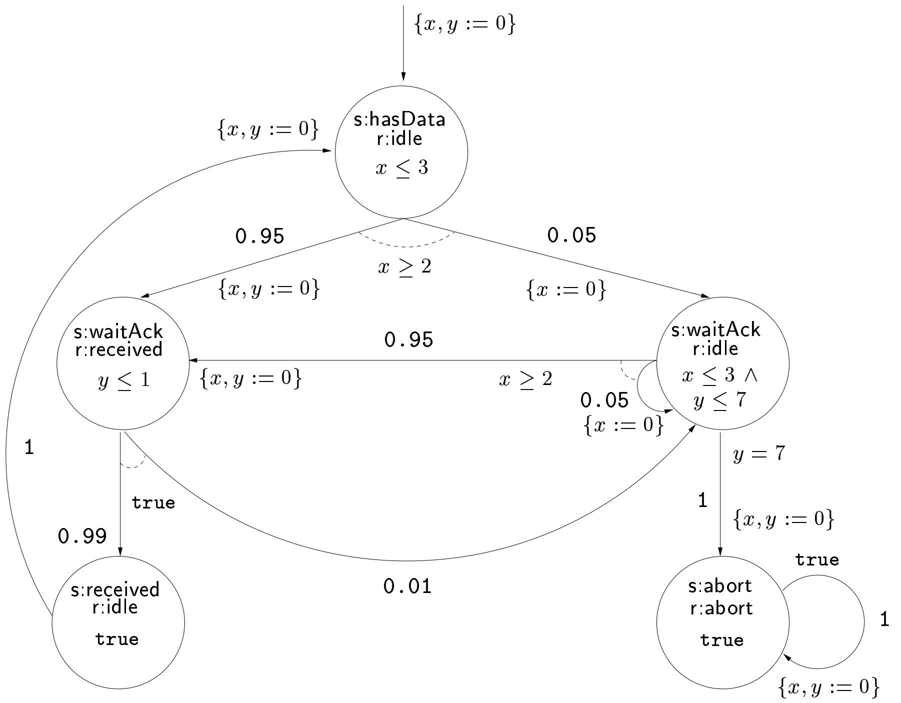


Figure 4.5: The probabilistic timed automaton T_{comm}.

description of probabilistic timed automaton behaviour in the nodes hi and wi, the behaviour of T_{comm} in other nodes follows in a straightforward manner. The reader can verify that T_{comm} does indeed model the communication protocol described in the previous paragraph. In subsequent sections, we will refer back to this example, using the abbreviations ri for s : received, r : idle, and aa for s : abort, r : abort.

In addition to considering the way in which the communication protocol can be modelled, we also identify a set of requirements which we wish the system to satisfy, and formalise them in an appropriate manner. Their validity with respect to the probabilistic timed automaton T_{comm} could then be verified using the model checking techniques presented in the remainder of this thesis (although, as we will see later, not all of these properties can be verified by both of our presented approaches). Four types of requirements are now presented, along with examples relevant to the communication protocol. Note that all of these properties are *probabilistic* variants of requirements that have appeared in the non-probabilistic, timed literature (see, for example, [DOTY96]).

Reachability The system can reach a certain set of states with a given probability. For example, "with probability 0.999 or greater, it is possible to correctly deliver a data packet".

Time bounded reachability The system can reach a certain set of states within a certain time deadline with a given probability. For example, "with probability

0.975 or greater, it is possible to correctly deliver a data packet within 5 time units"

Invariance The system does not leave a certain set of states with a given probability. For example, "with probability 0.875 or greater, the system never aborts".

Bounded response The system inevitably reaches a certain set of states within a certain time deadline with a given probability. For example, "with probability 0.99 or greater, a data packet will always be delivered within 5 time units".

In Chapters 5 and 6, we will see how each of these properties may be expressed in formalisms for which probabilistic timed automaton verification is possible.

4.2.4 Well-formedness assumptions

We now present two well-formedness assumptions on the syntax of probabilistic timed automata. The following assumption concerning the initial states of the model is made mainly for convenience, whereas the subsequent assumption referring to the preconditions of the model is made to prevent the situation in which a discrete transition is made to an inadmissible state.

Initial states

We identify the *initial states* of the probabilistic timed automaton T, which are defined by its initial condition $init$. Recall that, for any location $v \in V$, the initial condition is either a point in $\mathbb{N}^n$ (which is also a point in $\mathbb{R}^n$, and is therefore a valuation), or the empty set $\emptyset$. Then, for each location $v \in V$ for which $init(v) \neq \emptyset$, there exists an initial state $(v, init(v))$. We adopt the standard interpretation of multiple initial states which assumes that the system decides amongst the set according to a nondeterministic choice when commencing execution.

This definition may lead to the pathological situations in which an initial state is not admissible (the valuation given by $init(v)$ for a location $v \in V$ is such that $init(v) \notin inv(v)$), and that in which there is no initial state ($init(v) = \emptyset$ for all $v \in V$). We now address these problems, and also simplify the notion of initial condition, by forcing this condition to take a restricted form. Note that, for a probabilistic timed automaton which does not exhibit the above pathological characteristics, we can construct another probabilistic timed automaton with a *single* initial state without loss of generality; informally speaking, all that is required is that an additional location v' is added, at which no time is allowed to elapse, and which is associated with a number of Dirac distributions. Each of these Dirac distributions corresponds to an initial state, have the trivial enabling conditions of `true`, and is defined over a tuple $(v, init(v), \mathcal{X})$, which gives the location of that initial state, and sets all of the clocks to the required initial condition $init(v)$. Therefore, we assume that every probabilistic timed automaton T featured in the sequel is such that there exists a *single* location $v \in V$ which is such that $init(v) \neq \emptyset$. Furthermore, we also assume that $init(v) \in inv(v)$. This guarantees that there exists some (possibly finite) behaviour of the probabilistic timed automaton. The pathological situations which could subsequently arise are dealt with in later sections.

Tightening the pre-conditions

Observe the following two facts. First, it is not necessarily the case that, for any location $v \in V$ and distribution $p \in prob(v)$, the pre-condition of this distribution $pre_v(p)$ is contained within the invariant condition of the location $inv(v)$. Secondly, and more significantly, it may not be the case that a discrete transition of a probabilistic timed automaton T is made to an admissible state. Consider the location $v \in V$ and the distribution $p \in prob(v)$ which is such that $p(v', post) > 0$, for some location $v' \in V$ and post condition $post \in (\mathbb{N} \cup \{\perp\})^n$. Then it may not be the case that $pre_v(p)[X := post] \subseteq inv(v)$; in other words, there may exist a valuation $\mathbf{a} \in pre_v(p)$ such that $\mathbf{a}[X := post] \notin inv(v)$. We show how to 'tighten' the pre-conditions of any probabilistic timed automaton to obtain another probabilistic timed automaton for which the above possibilities are precluded.

Definition 4.2.2 (Tightened pre-conditions) *For a probabilistic timed automaton T, any location $v \in V$ and distribution $p \in prob(v)$ with the pre-condition $pre_v(p)$, the* tightened pre-condition *$pre'_v(p)$ of p is the zone:*

$$pre'_v(p) = pre_v(p) \cap inv(v) \cap \bigcap_{(v',post)\in \mathsf{support}(p)} [X := post]inv(v') \ .$$

To understand the intuitive meaning of this definition, consider a particular location $v \in V$ and distribution $p \in prob(v)$. Then $[X := post]inv(v')$ denotes the set of all valuations which result in a valuation in $inv(v')$ when clocks in the set X are reset according to the values in $post$, for a tuple $(v', post) \in \mathsf{support}(p)$. Taking the intersection of these sets of valuations for each tuple in $\mathsf{support}(p)$, along with the invariant set of the source location $inv(v)$ and the pre-condition $pre_v(p)$, results in the tightened pre-condition $pre'_v(p)$. Therefore, the set $pre'_v(p)$ of valuations are those for which, whatever the probabilistic choice made according to the distribution p, the resulting valuation must be contained within the appropriate invariant condition. Consider the following example: assume that $inv(v)$ is the trivial clock constraint `true`, and say $pre_v(p)$ is given by the clock constraints $x \geq 1 \wedge x \leq 3 \wedge y \geq 1 \wedge y \leq 3 \wedge x \geq y$. Furthermore, let $\mathsf{support}(p) = \{(v'_1, post_1), (v'_2, post_2)\}$, where the value of $post_1 = (0, \perp)$, $post_2 = (\perp, 0)$, and let $inv(v'_1) = x \leq 4 \wedge y < 2$, $inv(v'_2) = x \leq 3 \wedge y \leq 3$. We can then compute that $[\{x\} := post_1]inv(v'_1) = y < 2$ and $[\{y\} := post_2]inv(v'_2) = x \leq 3$.

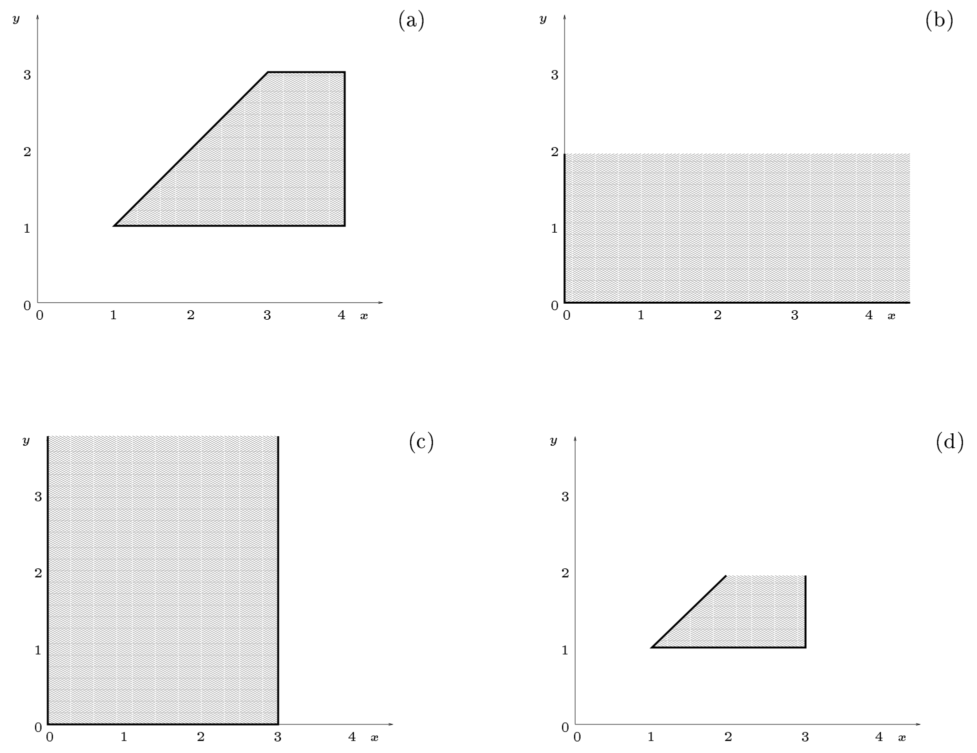


Figure 4.6: (a) The pre-condition $pre_v(p)$. (b) The set $[\{x\} := post_1]inv(v'_1)$. (c) The set $[\{y\} := post_2]inv(v'_2)$. (d) The tightened pre-condition $pre'_v(p)$.

Then the zones for $pre_v(p)$, $[\{x\} := post_1]inv(v'_1)$, and $[\{y\} := post_2]inv(v'_2)$ are given in Figure 4.6(a), (b) and (c), respectively. Note that a bold line represents a non-strict constraint, whereas an open edge of a zone corresponds to a strict constraint. We obtain the tightened pre-condition $pre'_v(p)$, as shown in figure Figure 4.6(d), by intersection of these three zones. Naturally, $pre'_v(p)$ can be represented by the conjunction of triangular clock constraints $x \leq 3 \wedge y \geq 1 \wedge y < 2 \wedge x \geq y$, which is equal to the conjunction of the clock constraints given above.

Henceforth, we assume that the pre-conditions of all of the locations of probabilistic timed automata are tightened.

4.3 Semantics of probabilistic timed automata

In this section, we explain the way in which concurrent probabilistic systems can be used as underlying semantic models for probabilistic timed automata. Intuitively, given a probabilistic timed automaton T, a state of the underlying concurrent probabilistic system $\mathcal{S}_T$ corresponds to a location of T plus a valuation of all of its clocks, whereas a transition of $\mathcal{S}_T$ corresponds to either a time transition or a probabilistic discrete transition of T. Furthermore, we model time transitions by Dirac distributions in the concurrent probabilistic system, and probabilistic transitions by more general distributions. The definition of adversaries of a probabilistic timed automaton then follows from that of adversaries of the underlying system $\mathcal{S}_T$, although we concentrate on a subset of adversaries that reflect natural assumptions about *progress* of both discrete and time transitions in Section 4.4.

Firstly, we formalise the notion of a probabilistic timed automaton *states* which were introduced in Section 4.2.2.

Definition 4.3.1 (States of a probabilistic timed automaton) *A* state *of a probabilistic timed automaton T is a pair $(v, \mathbf{a})$, where $v \in V$ is a location, and $\mathbf{a} \in \mathbb{R}^n$ is a clock valuation such that $\mathbf{a} \in inv(v)$.*

The initial state *of a probabilistic timed automaton T is the state $(v, init(v))$, where the location $v \in V$ is such that $init(v) \neq \emptyset$.*

To denote the initial location of the probabilistic timed automaton T, we use the symbol $\bar{v}$; formally, we let $\bar{v} \in V$ be the location for which $init(\bar{v}) \neq \emptyset$. As the behaviour of T commences in location $\bar{v}$ with its clocks set to values given by $init(\bar{v})$, our intuition is that the initial state $\bar{s}$ of the concurrent probabilistic system $\mathcal{S}_T$ is $(\bar{v}, init(\bar{v}))$. States of the probabilistic timed automaton T are also states of the underlying concurrent probabilistic system $\mathcal{S}_T$, and therefore we often denote a state of T simply as s, with the state space denoted by S_T (we occasionally elide the subscript when T is clear from the context). We use $\mathsf{discrete}(s)$ to denote the discrete part, or location, of s; that is, $\mathsf{discrete}(s) = v$ if $s = (v, \mathbf{a})$ for some $\mathbf{a} \in \mathit{inv}(v)$. Furthermore, to denote the passage of time of duration $\delta \in \mathbb{R}_{\geq 0}$ from a state $s = (v, \mathbf{a})$, we write $s + \delta$ for the state $(v, \mathbf{a} + \delta)$.

The dichotomous classification of probabilistic timed automaton transitions presented in Section 4.2.1 is reflected in the transition relation of the underlying semantic concurrent probabilistic system. That is, the distributions available for nondeterministic choice in a state s (which comprise the set $\mathit{Steps}(s)$) can be categorised as corresponding either to time transitions or probabilistic discrete transitions.

- A *time transition* of duration $\delta \in \mathbb{R}_{\geq 0}$ from the state s is represented by the Dirac distribution $\mathcal{D}(s + \delta)$ in the set $\mathit{Steps}(s)$. Recall that our intuition is that, for a given duration, the target state of a time transition is determined, and therefore the distribution relating to the passage of time is Dirac in nature. [1]

- A *probabilistic discrete transition* corresponding to the probabilistic timed automaton distribution p is represented by the distribution $\tilde{p}$ in the set $\mathit{Steps}(s)$. Intuitively, $\tilde{p}$ assigns to the state s' the *total* probability of making a transition to s' from s via p. That is, there may be more than one tuple $(v', \mathit{post}) \in \mathsf{support}(p)$ which corresponds to a transition from s to s'. This characteristic arises because, for any two valuations $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$, there may be more than one post condition post and clock set $X \subseteq \mathcal{X}$ such that $\mathbf{a}[X := \mathit{post}] = \mathbf{b}$. Therefore, the probability of each tuple in $\mathsf{support}(p)$ which corresponds to a transition from s to s' must be summed to obtain the probability $\tilde{p}(s')$.

Definition 4.3.2 (Semantics of a probabilistic timed automaton) *Let* $T = (\mathcal{X}, V, \mathcal{L}, \mathit{init}, \mathit{inv}, \mathit{prob}, \langle \mathit{pre}_v \rangle_{v \in V})$ *be a probabilistic timed automaton. Then the* concurrent probabilistic system $\mathcal{S}_T = (S, \bar{s}, L, \Sigma, \mathit{Steps})$ of T *is defined as follows:*

- S *is the set of states of* T.

- $\bar{s} = (\bar{v}, \mathit{init}(\bar{v}))$ *is the initial state of* T.

- *For each* $s \in S$*, the atomic propositions assigned to* s *by the labelling function* L *is* $L(s) = \mathcal{L}(\mathsf{discrete}(s))$.

- $\Sigma = \{\theta_p \mid p \in \mathit{prob}(v), v \in V\} \cup \mathbb{R}_{\geq 0}$ *is the event set.*

- *For each state* $(v, \mathbf{a}) \in S$*, let* $\mathit{Steps}(v, \mathbf{a}) = \mathit{Time}(v, \mathbf{a}) \cup \mathit{Disc}(v, \mathbf{a})$ *be the smallest set of event-distribution pairs such that:*

  - *for each duration* $\delta \in \mathbb{R}_{\geq 0}$*, there exists* $(\delta, \mathcal{D}(v, \mathbf{a} + \delta)) \in \mathit{Time}(v, \mathbf{a})$ *if and only if* $\mathbf{a} + \delta \in \mathit{inv}(v)$*;*

[1] This assumption is also made in the case of the probabilistic timed automata of [Seg95].

– *for each distribution p of T, there exists an event-distribution pair $(\theta_p, \tilde{p}) \in$ $Disc(v, \mathbf{a})$, if and only if $p \in prob(v)$, $\mathbf{a} \in pre_v(p)$, and $tp \in \mu(S)$ is such that, for any state $(v', \mathbf{b}) \in S$:*

$$\tilde{p}(v', \mathbf{b}) = \sum_{\substack{post \in (\mathbb{N} \cup \{\perp\})^n \; \& \\ \mathbf{b} = \mathbf{a}[X := post]}} p(v', post) \ .$$

We note that the event set comprises the set of all non-negative time durations $\mathbb{R}_{\geq 0}$ and events θ_p for each distribution $p \in prob(v), v \in V$ of the probabilistic timed automaton T. Note that θ_p is a generic name for a discrete transition, and encodes the fact that the distribution p that was used to make the transition.

Observe that, for any probabilistic timed automaton T, the concurrent probabilistic system $\mathcal{S}_T$ is a pointed concurrent probabilistic system (the initial state $\bar{s}$ is unique). Transitions are denoted in the following manner. Firstly, a time transition of duration $\delta \in \mathbb{R}_{\geq 0}$ from a state s is denoted $s \xrightarrow{\delta} s + \delta$, rather than by $s \xrightarrow{\delta, \mathcal{D}(s+\delta)} s + \delta$, as, given s and δ, the information that the transition corresponds to the Dirac distribution $\mathcal{D}(s+\delta)$ is clear. Similarly, it will usually be convenient to denote a probabilistic transition from a state s to a state s' by $s \xrightarrow{\tilde{p}} s'$, rather than by $s \xrightarrow{\theta_p, \tilde{p}} s'$. The information encoded in the event θ_p (that the concurrent probabilistic system event-distribution pair $(\theta_p, \tilde{p})$ is derived from the probabilistic timed automaton distribution p) is generally irrelevant, and will be used mainly for proofs later in the thesis. Furthermore, when the time transition $s \xrightarrow{\delta} s + \delta$ is followed by the probabilistic transition $s + \delta \xrightarrow{\tilde{p}} s'$, we often write $s \xrightarrow{\delta} \xrightarrow{\tilde{p}} s'$.

Paths of a semantic concurrent probabilistic system $\mathcal{S}_T$ of a probabilistic timed automaton T, comprising a finite or infinite sequence of time and probabilistic transitions, are defined in the standard way.

Our notion of real-valued time, combined with the fact that the choice of the successor of a time transition of a certain duration is deterministic, allows us to state the following facts. Firstly, the two time transitions $s \xrightarrow{\delta} s + \delta$ and $s + \delta \xrightarrow{\delta'} s + \delta + \delta'$ are equivalent to the time transition $s \xrightarrow{\delta + \delta'} s + \delta + \delta'$. We call this the *concatenation property* of time transitions. Secondly, a time transition $s \xrightarrow{\delta} s + \delta$ can be split into two time transitions $s \xrightarrow{\delta'} s + \delta'$ and $s + \delta' \xrightarrow{\delta - \delta'} s + \delta$, for some $0 \leq \delta' \leq \delta$. We call this *time additivity* [NS91]. Furthermore, by reasoning inductively, the former property implies that an arbitrary number of consecutive time transitions can be concatenated into a single time transition, while the second property implies that a single time transition can be split into a sequence of time transitions of arbitrary length.[2]

Occasionally, it will be desirable to abstract from the exact duration of time transitions. In particular, the relations of simulation and bisimulation with respect to *exact*

[2]The second property has its basis in the work of Wang Yi [Yi90] (and is sometimes called "Wang's axiom"), and [NS91].

time durations may be too fine (that is, they may distinguish between states too much for our purposes). Therefore, we introduce the time-abstract concurrent probabilistic system of a probabilistic timed automaton T with respect to a PTCTL formula. Such a system is simply obtained from the concurrent probabilistic system of T by replacing all time events (those with a duration in $\mathbb{R}_{\geq 0}$) with the 'time-abstracting' event τ.

Definition 4.3.3 *The* time-abstract concurrent probabilistic system $\mathsf{ta}(\mathcal{S}_T) = (S, \bar{s}, L, \Sigma', \mathit{Steps}')$ *of the probabilistic timed automaton T is defined according to Definition 4.3.2 and as follows:* $\Sigma' = \{\{\theta_p \mid p \in \mathit{prob}(v), v \in V\} \cup \tau\}$ *and, for every state* $(v, \mathbf{a}) \in S$, *we have* $\mathit{Steps}(v, \mathbf{a}) = \mathit{Time}'(v, \mathbf{a}) \cup \mathit{Disc}(v, \mathbf{a})$, *where there exists* $(\tau, \mathcal{D}(v, \mathbf{b})) \in \mathit{Time}'(v, \mathbf{a})$ *if and only if there exists* $(\delta, \mathcal{D}(v, \mathbf{b})) \in \mathit{Time}(v, \mathbf{a})$ *for some* $\delta \in \mathbb{R}_{\geq 0}$.

4.4 Progress in probabilistic timed automata

Given the description of the semantics of probabilistic timed automata in Section 4.3, we observe the following two facts. Firstly, it is possible for the underlying concurrent probabilistic system $\mathcal{S}_T$ to exhibit an infinite path in which only a finite number of probabilistic discrete transitions are taken. That is, a probabilistic timed automaton can choose not to make any probabilistic choice, and therefore choose not to make possible progress amongst its discrete states, in some execution. Indeed, there may be paths in which a discrete transition is *not possible* from some point onwards; such a phenomenon is called a *deadlock*. Secondly, it is possible for $\mathcal{S}_T$ to exhibit a path in which the total time duration converges to a finite value. In other words, a probabilistic timed automaton can choose to let time elapse in such a way so that the total time passed since the commencement of system execution does not progress beyond a certain bound. As for the case of deadlocks, there may be paths in which such convergence of time is unavoidable, a phenomenon known as a *timelock*. Both of these characteristic do not tally with out 'reactive' view of probabilistic timed automata, which interprets the model as having an ongoing interaction with its environment.

We therefore introduce dynamic and static ways of studying the issues of progress of probabilistic timed automata. The dynamic approach concerns the definition of progress over infinite paths and adversaries, whereas the static approach considers guarantees of deadlock-freedom by consideration of the syntactic structure of the probabilistic timed automaton. We leave the probabilistic analogue of the work of Tripakis [Tri98, Tri99b] for static guarantees of timelock-freedom for future work.

4.4.1 Dynamic methods for progress

In this section, we concentrate attention on behaviours of probabilistic timed automata which make both discrete and timed progress. More precisely, we consider adversaries

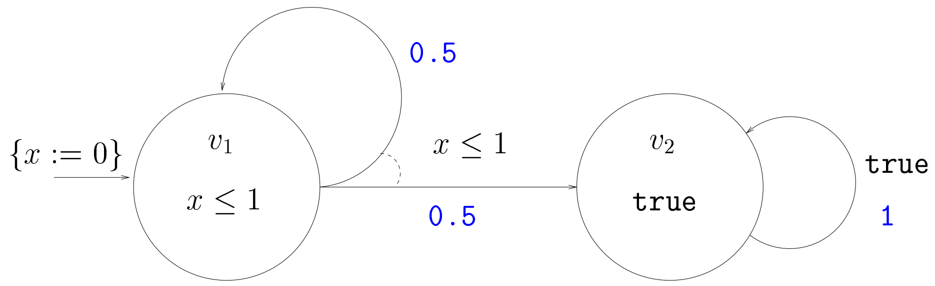


Figure 4.7: The probabilistic timed automaton T_{time}

which exhibit discrete and timed progress *with probability 1*. The justification for this requirement of *probabilistic progress*, rather than the requirement of discrete and timed progress over *all* paths of an adversary, takes the following form. Firstly, the requirements of discrete progress with probability 1, and that of discrete progress along all of an adversary's paths, are equivalent. Suppose that, for the adversary A, we have a path $\omega \in Path^A_{ful}$ such that there are only finitely many transitions of the form $s \xrightarrow{\tilde{p}} s'$. Then, from the fact that all other transitions are time transitions, and as such are made with probability 1, it must be the case that $Prob^A(\omega) > 0$. Therefore, if a path features a finite number of probabilistic transitions that correspond to actual transitions in the graph of the probabilistic timed automaton, its measure must be nonzero.

Hence, we define infinite paths exhibiting *discrete progress* to be those that feature an infinite sequence of alternating time and probabilistic transitions. The properties of concatenation of time transitions and time additivity of Section 4.3 allow us to concentrate our attention on the class of paths which alternate between the choice of time transitions and probabilistic transitions when considering discrete progress. Firstly, observe that a sequence of transitions of the form $s \xrightarrow{\tilde{p}} s' \xrightarrow{\tilde{p}'} s''$ can be transformed into the sequence $s \xrightarrow{\tilde{p}} s' \xrightarrow{\delta} s' \xrightarrow{\tilde{p}'} s''$, for $\delta = 0$. Furthermore, a sequence of the form $s \xrightarrow{\delta_1} s + \delta_1 \xrightarrow{\delta_2} s + \delta_1 + \delta_2 \xrightarrow{\delta_3} \dots \xrightarrow{\delta_m}\xrightarrow{\tilde{p}} s'$, for some finite $m \in \mathbb{N}$, can be transformed into the sequence $s \xrightarrow{\sum_{i=1}^{m} \delta_i}\xrightarrow{\tilde{p}} s'$. Therefore, it is sufficient to study paths for which time transitions and probabilistic transitions alternate. If we consider infinite paths of this form, then they must feature infinitely many time transitions, and, more importantly in the context of discrete progress, infinitely many probabilistic transitions. This inspires the following definition.

Definition 4.4.1 (Paths with discrete progress) *Let T be a probabilistic timed automaton. A path $\omega \in Path_{ful}$ of the concurrent probabilistic system $\mathcal{S}_T$ of T exhibits* discrete progress *if it is of the form $\omega = s_0 \xrightarrow{\delta_0}\xrightarrow{\tilde{p}_0} s_1 \xrightarrow{\delta_1}\xrightarrow{\tilde{p}_1} s_2 \xrightarrow{\delta_2}\xrightarrow{\tilde{p}_2} \dots$.*

Secondly, to see that the requirements of probabilistic time divergence and the progress of time beyond any bound along *all* paths of an adversary should be distinguished, consider the following example. Observe that *every adversary* of the underlying system of the probabilistic timed automaton in Figure 4.7 exhibits a path of the form

$$\omega_{\text{time}} = (v_1, \mathbf{0}) \xrightarrow{\delta_0}\xrightarrow{\tilde{p}} (v_1, \mathbf{0} + \delta_0) \xrightarrow{\delta_1}\xrightarrow{\tilde{p}} (v_1, \mathbf{0} + \delta_0 + \delta_1) \xrightarrow{\delta_2}\xrightarrow{\tilde{p}} \ldots$$

such that $\sum_{i=0}^{\infty} \delta_i \leq 1$. Clearly, ω_{time} is not a time-divergent path; however, consider the probability of its occurrence. Observe that the probability of the infinite set of infinite paths for which the edge from v_1 to v_2 is traversed on the ith transition, for some $i \geq 1$, is equal to $(0.5)^i$. That is, there exists a multiple in this product for each of the first i transitions along the finite prefix of all paths in this set; $i-1$ of these multiples correspond to the traversal of the self-loop edge of v_1 on the first $i-1$ transitions, while the ith multiple corresponds to the traversal of the edge from v_1 to v_2 on the ith transition. Also note that such a set of paths is a basic cylinder (see Section 3.2.1). As all countable unions of basic cylinders are in the sigma-algebra $\mathcal{F}_{Path}$, we note that the union of the set of paths for which the edge from v_1 to v_2 is taken on the ith transition, for *any* $i \geq 1$, is also in the sigma-algebra $\mathcal{F}_{Path}$, and is therefore measurable. Then the probability of this set can be obtained by taking the sum of the probabilities of the basic cylinders described above, and so will be $\sum_{i \geq 1} (0.5)^i = 1$. Then, the path for which there does not exist an $i \geq 1$ such that the edge from v_1 to v_2 is traversed on the ith transition will have probability 0.

Now consider the case in which we require that all paths of an adversary are time-divergent. Then any adversary exhibiting such a path would be regarded as being inadmissible, and thus the underlying structure of the probabilistic timed automaton in Figure 4.7 would have no admissible adversaries, violating our intuition that control should be able to pass from location v_1 to location v_2 with probability 1, for all adversaries. Therefore, we regard this as an overly strong definition, and instead take the view that such pathological behaviour as time-convergence to be acceptable if it has probability 0.

To formalise our notation of the duration of time elapsed along a path, we introduce the following function.

Definition 4.4.2 (Duration of a path) *For any path* $\omega = s_0 \xrightarrow{\sigma_0} s_1 \xrightarrow{\sigma_1} \ldots$ *of the concurrent probabilistic system* $\mathcal{S}_T$ *of the probabilistic timed automaton* T*, and* $0 \leq i \leq |\omega|$*, we define* $\text{Dur}_\omega(i)$*, the* elapsed time *until the ith transition, as follows:* $\text{Dur}_\omega(0) = 0$ *and for any* $1 \leq i \leq |\omega|$*:*

$$\text{Dur}_\omega(i) = \sum_{j<i \,\&\, \sigma_j \in \mathbb{R}_{\geq 0}} \sigma_j \, .$$

This permits us to introduce the following definition of time progress along paths.

Definition 4.4.3 (Paths with time progress) *A path $\omega \in Path_{ful}$ of $\mathcal{S}_T$ exhibits* time progress *if, for any $\delta \in \mathbb{R}_{\geq 0}$, there exists $i \in \mathbb{N}$ such that $\mathrm{Dur}_\omega(i) > \delta$.*

We now introduce three special classes of adversaries. *Discrete-progress adversaries* reflect the discrete progress requirement, and define infinite paths featuring alternating time and probabilistic transitions. *Time-progress adversaries* are defined with respect to the time divergence requirement. Finally, we combine the two adversary types to define *divergent* adversaries. Consider the probabilistic timed automaton T and its associated semantic concurrent probabilistic system $\mathcal{S}_T$.

Definition 4.4.4 (Discrete-progress adversaries) *An adversary $A \in Adv$ of $\mathcal{S}_T$ is a* discrete-progress adversary *if and only if each path $\omega \in Path^A_{ful}$ exhibits discrete progress.*

Definition 4.4.5 (Time-progress adversaries) *An adversary $A \in Adv$ of $\mathcal{S}_T$ is a* time-progress adversary *if and only if*

$$Prob^A\{\omega \in Path^A_{ful} \mid \omega \text{ exhibits time progress}\} = 1.$$

The probability defined in Definition 4.4.5 is well-defined for the following reasons. First observe that Definition 4.4.3 is equivalent to an identical definition with $\delta \in \mathbb{R}_{\geq 0}$ replaced with $\delta \in \mathbb{N}$. Then, for a natural number $\delta \in \mathbb{N}$, consider the set of paths of an adversary A for which the total time elapsed is greater than δ. It follows that such a set is an open set. Now consider the countable intersection of such sets, for all $\delta \in \mathbb{N}$. Such a countable intersection of open sets is in the sigma-algebra defined in Section 3.2.1, and is therefore measurable.

We now combine the characteristics of discrete-progress and time-progress to obtain divergent adversaries.

Definition 4.4.6 (Divergent adversaries) *An adversary $A \in Adv$ of $\mathcal{S}_T$ is a* divergent adversary *if and only if it is a discrete-progress adversary and a time-progress adversary. Let Div be the set of divergent adversaries of $\mathcal{S}_T$.*

4.4.2 Static methods for progress

We now introduce two notions of *well-formedness with respect to progress* of probabilistic timed automata, which take the form of assumptions on the syntactic structure of the model. Although we generally assume one of these static conditions for technical reasons, the absence of either corresponds to modelling errors which could cause deadlocks, which, as we have stated, do not correspond to the reactive view of probabilistic timed automata.

Firstly, we define the well-formedness property of *non-deadlocking invariants.* To facilitate the definition, we introduce the following notation.

Definition 4.4.7 *Let $\zeta \in \mathsf{Zone}(\mathcal{X})$ be a zone, and consider its syntactic representation as a conjunction of atomic formulae in canonical form. Then, we let $\zeta' \in \mathsf{Zone}(\mathcal{X})$ be the zone obtained from ζ by the following substitutions:*

- *for each atomic formula of ζ of the form $x \leq c$, where $x \in \mathcal{X}$ and $c \in \mathbb{N}$, we substitute the atomic formula $x < c$; and*
- *for each atomic formula of ζ of the form $x < c$, where $x \in \mathcal{X}$ and $c \in \mathbb{N}$, we substitute the atomic formula $x \leq (c-1)$.*

Then we let $\mathsf{ubound}(\zeta) \in \mathsf{Poly}(\mathcal{X})$ be the possibly non-convex polyhedron $\zeta \setminus \zeta'$.

Intuitively, non-deadlocking invariants require that the states in which the passage of time will violate each invariant are within at least one pre-condition. We formalise this notion in the following definition.

Definition 4.4.8 (Non-deadlocking invariants) *A probabilistic timed automaton T has* non-deadlocking invariants *if and only if* $\mathsf{ubound}(inv(v)) \subseteq \bigcup_{p \in prob(v)} pre_v(p)$ *for all locations $v \in V$.*

The presence of non-deadlocking invariants means that, for *any* admissible state $(v, \mathbf{a}) \in S$ of $\mathcal{S}_T$, if advancing time by some positive but not infinitesimal duration $\delta > 0$ would violate the invariant condition (that is, it is the case that $\mathbf{a} + \delta \notin inv(v)$), then there must exist an enabled distribution (that is, we have $\mathbf{a} \in pre_v(p)$ for some $p \in prob(p)$).

As the notion of non-deadlocking invariants is rather strong, we weaken it to define *sufficient deadlock freedom* in the style of [Tri98, Tri99b]. We now identify the states which both satisfy the invariant and a pre-condition, and all of the time predecessors of these states. For each $v \in V$, let $\mathsf{free}(v) \stackrel{def}{=} \bigcup_{p \in prob(v)} \swarrow pre_v(p)$ [3]. Sufficient deadlock freedom requires that, for any edge $e \in E$ which has the location v as its target, the edge e also has a pre-condition and a reset set such that, for any source state which can take e, the resulting target state is in $\mathsf{free}(v)$. The initial state is a special case which follows naturally.

Definition 4.4.9 (Sufficient deadlock freedom) *A probabilistic timed automaton T has* sufficient deadlock freedom *if and only if:*

- *for the initial location $\bar{v}$, we have $init(\bar{v}) \in \mathsf{free}(\bar{v})$; and*
- *for every other location $v \in V \setminus \{\bar{v}\}$, and for each $(v', v, post, p) \in \mathsf{in}(v)$, we have $(pre_{v'}(p) \cap inv(v'))[X := post] \cap inv(v) \subseteq \mathsf{free}(v)$.*

[3]This definition differs slightly from that of Tripakis [Tri98, Tri99b], as we have assumed that the probabilistic timed automaton in question has tightened pre-conditions.

Observe that sufficient deadlock freedom is weaker than non-deadlocking invariants. If a probabilistic timed automaton has non-deadlocking invariants, then it also has sufficient deadlock freedom, but not necessarily the reverse.

Chapter 5

Model checking temporal logic properties

5.1 Introduction

This chapter presents a model checking technique for the automatic verification of probabilistic timed automata against temporal logic specifications. Our approach is a synthesis of the *region graph* based technique for model checking timed automata [AD94, ACD93], and the automatic verification algorithm of [BdA95, BK98] for concurrent probabilistic systems. The new verification method is with respect to properties of a real-time temporal logic which can refer to bounds on the likelihood of satisfaction, meaning that we can express such properties as:

- "with probability 0.99 or greater, a non-critical malfunction does not occur within 30 minutes" (a probabilistic, timed safety property);
- "with probability 0.95 or greater, the system user receives a warning within 5 milliseconds after the occurrence of a malfunction" (a probabilistic, timed response property);
- "with probability strictly greater than 0.999, if a malfunction has occurred, a shutdown will be implemented within 1 minute, and the system is in the mode alert until this shutdown has occurred" (a probabilistic, timed response property using an 'until' temporal modality).

Recall from Section 2.3.1 that the dense time domain of timed automata, and their variants such as probabilistic timed automata, means that their state space is infinite, thereby preventing direct use of model checking techniques for finite-state systems, as pioneered by Clarke et al. [CES86]. However, for any timed automaton, *region equivalence* [AD94] is a finite equivalence relation on the set of states, and, as such, induces a finite partition on the state space, which can then be used to obtain a finite-state quotient transition system, called the *region graph*, The simplicity of the region graph approach is based on the fact that region equivalence is primarily defined *without* respect to the transition structure of a particular timed automaton: the state space is partitioned in a regular manner for all timed automata, with limits on the partitioning being given by the maximal integer constant used in the system description (that is, in a zone used to define the model's invariant and enabling conditions).

We show in this chapter that a literal application of region equivalence to the state space of probabilistic timed automata results in an equivalence relation which preserves formulae of a probabilistic, timed temporal logic. More precisely, the behaviours leaving each state in a region equivalence class are considered to be identical by our logic. The intuition underlying this result is similar to that in the case of timed automata and the logic TCTL [ACD93]: every state in a given region equivalence class must satisfy the same formulae on clocks, which then means that they must satisfy currently the same enabling or invariant conditions. We can also show that, as time passes, the same enabling or invariant conditions are satisfied. Therefore, such states must enable the same set of discrete transitions for choice, or make sufficiently similar time transitions.

The temporal logic presented here is called Probabilistic Timed Computation Tree Logic (PTCTL), and synthesises elements from both probabilistic and real-time extensions of the branching time logic CTL. In particular, the temporal operator $\mathcal{U}$ ("until") and the path quantifiers $\forall$ and $\exists$ ("for all" and "there exists", respectively) are taken from CTL [CE81], whereas the reset quantifier $z.\phi$ and the facility to refer directly to clock values are taken from TCTL [ACD93, HNSY94] (more precisely, the version of TCTL presented in [HNSY94]). Finally, the probabilistic operators $[\phi_1 \exists \mathcal{U} \phi_2]_{\sqsupseteq \lambda}$ and $[\phi_1 \forall \mathcal{U} \phi_2]_{\sqsupseteq \lambda}$, which permit reference to bounds on the likelihood of properties, are taken from the logics PCTL [HJ94, Bai98], pCTL [BdA95] and PBTL [BK98]. In particular, the probabilistic operators of the logic PTCTL has the most syntactic similarities with PBTL, as the quantification over adversaries is made explicit both logics, as opposed to being implicit in both PCTL and pCTL.

We present the syntax of PTCTL in Section 5.2, along with its semantics with respect to the concurrent probabilistic systems of probabilistic timed automata. The definition of region equivalence for probabilistic timed automata is given in Section 5.3. The subsequent section defines the resulting probabilistic region graph, and identifies the correspondences between the paths and adversaries of this finite-state model and those of the original probabilistic timed automaton. Finally, in Section 5.5, we use these correspondences to show that region equivalence preserves PTCTL properties, thereby justifying the use of a probabilistic-nondeterministic variant of the region graph in PTCTL model checking of probabilistic timed automata.

For simplicity, we restrict the class of probabilistic timed automata featured in this chapter to those for which the only value that clocks can be reset to in post conditions is 0, and for which the initial state is $(\bar{v}, \mathbf{0})$. This restriction is made without loss of generality, as the "folklore" result which state that, for any non-probabilistic timed automaton which resets clocks to integers, a bisimilar timed automaton for which clocks are reset to 0 only can be constructed (see Theorem 1 in [BDFP00b]), also applies to the probabilistic context. We then denote post conditions as the set of clocks which are reset to 0; for example, the post condition *post* is replaced by the set of clocks $X \subseteq \mathcal{X}$ for which $post_i = 0$ for each $x_i \in X$.

Note that material from this chapter first appeared in [KNSS99, KNSS00a].

5.2 Probabilistic Timed Computation Tree Logic

We now describe the probabilistic real-time logic PTCTL (Probabilistic Timed Computation Tree Logic) which can be used to specify properties of probabilistic timed systems. Unless specifically stated, we refer to the probabilistic timed automaton $T = (\mathcal{X}, V, \mathcal{L}, init, inv, prob, \langle pre_v \rangle_{v \in V})$ throughout this section.

5.2.1 Syntax of PTCTL

We proceed by first explaining informally how properties may be expressed in PTCTL. Then we introduce the additional class of clocks that can be used to express timing properties within PTCTL formulae, before describing formally the syntax of PTCTL.

Expressing properties in PTCTL

The interpretation of the atomic propositions, boolean connectives, until formulae $\mathcal{U}$ (and their associated syntactic abbreviations $\Box, \Diamond$), and probability bound operators $[\phi_1 \exists \mathcal{U} \phi_2]_{\sqsupseteq \lambda}$ and $[\phi_1 \forall \mathcal{U} \phi_2]_{\sqsupseteq \lambda}$, follows directly from the interpretation of their associated PBTL formulae of Section 3.3. The logic PTCTL is then obtained from PBTL by the addition of two syntactical elements: firstly, *zones* can be used as atomic elements in PTCTL formulae (recall the definition of zones in Section 3.4), and, secondly, the *reset quantifier* of TCTL [HNSY94] can be used as a unary operator in a PTCTL formula. Our interpretation of these two elements is as follows.

A zone $\zeta \in \mathsf{Zone}(\mathcal{X})$ is interpreted as being true in a probabilistic timed automaton state $(v, \mathbf{a})$ if and only if the clock valuation $\mathbf{a}$ satisfies the conjunction of formulae on clocks given by the syntactical interpretation of ζ. Invariably, when writing zones as subformulae of PTCTL formulae, it is more convenient to write the syntactical interpretation of a zone as a conjunction of formulae in canonical form.

Let z be an additional clock not in $\mathcal{X}$ called a *formula clock*, and ϕ be a PTCTL formula. Then the *reset quantifier* $z.\phi$ is true in a state $(v, \mathbf{a})$ if and only if ϕ is true in the state $(v, \mathbf{a}[z := 0])$, which is obtained from $(v, \mathbf{a})$ after resetting z to 0. The combination of such quantifiers and zones permits the expression of temporal logic properties such as, "a request can be followed by a response within 5 seconds with probability 0.99 or greater", which is represented by the PTCTL formula $[\forall\Box(z.\mathsf{request} \rightarrow [\forall\Diamond(\mathsf{response} \wedge z < 5)]_{\geq 0.99})]_{\geq 1}$ (note the use of the formula $z < 5$ as the syntactic representation of a zone).

The properties in Section 5.1 can be written as PTCTL formulae as follows:

- "with probability 0.99 or greater, a non-critical malfunction does not occur within 30 minutes" is written as

$$\neg z.[\exists\Diamond(\mathsf{malfunction} \wedge z < 30)]_{>0.01}$$

(it is not the case that there exists an adversary such that a malfunction occurs within 30 minutes with probability strictly greater than 0.01);

- "with probability 0.95 or greater, the system user receives a warning within 5 milliseconds after the occurrence of a malfunction" is written as:

$$[\forall\Box(z.\mathsf{malfunction} \rightarrow [\forall\Diamond(\mathsf{warning} \wedge z < 5)]_{\geq 0.95})]_{\geq 1}\,;$$

- "with probability strictly greater than 0.999, if a malfunction has occurred, a shutdown must be implemented within 1 minute, and the system is in the mode alert until this shutdown has occurred" is written as:

$$[\forall\Box(z.\mathsf{malfunction} \rightarrow [\mathsf{alert}\forall\mathcal{U}(\mathsf{shutdown} \wedge z < 60)]_{>0.999})]_{\geq 1}\,.$$

Similarly, we can write the properties of Section 4.2.3 in terms of PTCTL formulae. The property,"with probability 0.999 or greater, it is possible to correctly deliver a data packet", can be written as $[\exists\Diamond\mathsf{ri}]_{\geq 0.999}$. The PTCTL formula $\neg[\exists\Diamond\neg\mathsf{aa}]_{>0.0125}$ represents the property, "with probability 0.875 or greater, the system never aborts". The bounded reachability property, "with probability 0.99 or greater, a data packet will always be delivered within 5 time units", can be written as $z.[\forall\Diamond(\mathsf{ri} \wedge z \leq 5)]_{\geq 0.99}$.

Formula clocks

As with TCTL, the logic PTCTL employs a set of clock variables in order to express timing properties. For this purpose, we introduce *a set of formula clocks*, $\mathcal{Z}$, which is disjoint from $\mathcal{X}$. Take the probabilistic timed automaton T, with the clock set $\mathcal{X}$, and the PTCTL formula ϕ, which features zones defined by conjunctions of constraints referring to clocks from both $\mathcal{X} = \{x_1, ..., x_n\}$ and $\mathcal{Z} = \{z_1, ..., z_m\}$, and which features reset quantifiers referring to clocks in $\mathcal{Z}$ only. We then evaluate the formula ϕ over the probabilistic timed automaton $T^\phi = (\mathcal{X} \cup \mathcal{Z}, V, \mathcal{L}, init, inv, prob, \langle pre_v \rangle_{v \in V})$, which is identical to T except for the fact that its clock set is now $\mathcal{X} \cup \mathcal{Z}$.

Naturally, our view of valuations must be redefined in order to accommodate formula clocks. We overload notation by defining a valuation as a vector of length $n+m$ in $\mathbb{R}^{n+m}$, which assigns a real value to each clock $x_i \in \mathcal{X}$ for all $1 \leq i \leq n$, and $z_i \in \mathcal{Z}$ for all $1 \leq i \leq m$. Then our notion of a probabilistic timed automaton state now includes information concerning the clocks in the PTCTL formula ϕ.

From Definition 4.3.2, it is straightforward to derive the concurrent probabilistic system $\mathcal{S}_{T^\phi}$, which we henceforth write as $\mathcal{S}_T^\phi$ to ease notation. As formula clocks do not have any direct influence on the transitions of the concurrent probabilistic system (the definitions of invariant conditions, pre-conditions, clock resets and discrete transitions of T^ϕ still refer to clocks in $\mathcal{X}$ only), the behaviour of $\mathcal{S}_T^\phi$ is identical to that of $\mathcal{S}_T$ given that the values of the values of the formula clocks in $\mathcal{Z}$ are projected from the states of $\mathcal{S}_T^\phi$. However, the initial state of $\mathcal{S}_T^\phi$ will generally not be unique. If we denote by $\mathbf{0}_{|\mathcal{X}}$ a clock valuation assigning 0 to each of the clocks in $\mathcal{X}$ and an arbitrary value to all other clocks (that is, to the formula clocks in $\mathcal{Z}$), then the initial states of $\mathcal{S}_T^\phi$ will all be of the form $\mathbf{0}_{|\mathcal{X}}$.

Definition 5.2.1 (Concurrent probabilistic system of T^ϕ) *Let T be a probabilistic timed automaton, and let ϕ be a formula of PTCTL. The concurrent probabilistic system $\mathcal{S}_T^\phi = (S, \bar{S}, L, \Sigma, Steps)$ of T^ϕ is defined as follows:*

- *S is the set of states of T^ϕ.*
- *$\bar{S}$ is the set of initial states, each of which is of the form $(\bar{v}, \mathbf{0}_{|\mathcal{X}})$.*
- *L, Σ and Steps are defined as in Definition 4.3.2.*

Formal definition of PTCTL syntax

We now define formally the syntax of PTCTL.

Definition 5.2.2 (Syntax of PTCTL) *The syntax of PTCTL is defined as follows:*

$$\phi ::= \texttt{true} \mid a \mid \zeta \mid \phi \wedge \phi \mid \neg\phi \mid z.\phi \mid [\phi \,\exists\, \mathcal{U}\ \phi]_{\sqsupseteq\lambda} \mid [\phi \,\forall\, \mathcal{U}\ \phi]_{\sqsupseteq\lambda}$$

where $a \in$ AP is an atomic proposition, $\zeta \in \mathsf{Zone}(\mathcal{X} \cup \mathcal{Z})$ is a zone, $z \in \mathcal{Z}$ is a formula clock, $\lambda \in [0,1]$, and $\sqsupseteq \in \{\geq, >\}$.

For any PTCTL formula ϕ, we write $\mathsf{sub}(\phi)$ for the set of PTCTL subformulae of ϕ (we include ϕ itself within this set), and $\mathsf{Zone}(T, \phi)$ for the set of zones that are either in $\mathsf{Zone}(T)$ or appear in $\mathsf{sub}(\phi)$. Abbreviations of PTCTL formulae can be defined in exactly the same manner as similar abbreviations of PBTL formulae. The PTCTL formulae $[\texttt{true}\forall\mathcal{U}\Phi]_{\sqsupseteq\lambda}$ and $[\texttt{true}\exists\mathcal{U}\Phi]_{\sqsupseteq\lambda}$ can be written as $[\forall\Diamond\Phi]_{\sqsupseteq\lambda}$ and $[\exists\Diamond\Phi]_{\sqsupseteq\lambda}$ respectively. We also write the abbreviation $[\forall\Box\Phi]_{\sqsupseteq\lambda}$ for $\neg[\exists\Diamond\neg\Phi]_{\bar{\sqsupseteq}1-\lambda}$, and $[\exists\Box\Phi]_{\sqsupseteq\lambda}$ for $\neg[\forall\Diamond\neg\Phi]_{\bar{\sqsupseteq}1-\lambda}$, where $\bar{\geq} =>$ and $\bar{>} =\geq$.

5.2.2 Semantics of PTCTL

We now present the satisfaction relation for PTCTL with respect to the concurrent probabilistic system of a probabilistic timed automaton. Consider the probabilistic timed automaton T, and the PTCTL formula ϕ. Recall from Section 3.4 that, for a valuation $\mathbf{a} \in \mathbb{R}^{n+m}$ and a zone $\zeta \in \mathsf{Zone}(\mathcal{X} \cup \mathcal{Z})$, we write $\mathbf{a} \in \zeta$ to denote the fact that $\mathbf{a}$ is a member of the set of valuations defined by ζ. Equivalently, viewing ζ as a conjunction of atomic formulae over clocks of $\mathcal{X}$ and $\mathcal{Z}$, we have $\mathbf{a} \in \zeta$ if and only if substitution of the clock assignments of $\mathbf{a}$ for the appropriate clocks in ζ results in ζ resolving to `true`.

In the following, if the state $s \in S$ is also written as $(v, \mathbf{a})$, for a location $v \in V$ and a valuation $\mathbf{a} \in \mathbb{R}^{n+m}_{\geq 0}$, then $s[z := 0]$ denotes $(v, \mathbf{a}[z := 0])$. Furthermore, for a zone ζ, we write $s \in \zeta$ if and only if $\mathbf{a} \in \zeta$.

Consider an infinite path $\omega \in \mathit{Path}_{\mathit{ful}}$ of $\mathcal{S}^{\phi}_{T}$ which exhibits discrete progress; that is, it is of the form $\omega = s_0 \xrightarrow{\delta_0} \xrightarrow{\tilde{p}_0} s_1 \xrightarrow{\delta_1} \xrightarrow{\tilde{p}_1} s_2 \xrightarrow{\delta_2} \xrightarrow{\tilde{p}_2} \ldots$. A *position* of ω is a pair (i, δ), where $i \in \mathbb{N}$ and $\delta \in \mathbb{R}_{\geq 0}$ such that $0 \leq \delta \leq \delta_i$. The *state at position* (i, δ) is the state $s_i + \delta$. Given such an infinite path $\omega \in \mathit{Path}_{\mathit{ful}}$, $i, j \in \mathbb{N}$ and $\delta, \delta' \in \mathbb{R}$ such that $\delta \leq \delta_i$ and $\delta' \leq \delta_j$, then we say that the position (j, δ') *precedes* the position (i, δ), written $(j, \delta') \ll (i, \delta)$, if and only if $j < i$, or $j = i$ and $\delta' < \delta$.

Definition 5.2.3 (Satisfaction Relation for PTCTL) *Consider the probabilistic timed automaton T and the PTCTL formula ϕ. Then, for any state $s \in S$ of the concurrent probabilistic system $\mathcal{S}^{\phi}_{T} = (S, \bar{S}, L, \Sigma, \mathit{Steps})$, and a set $\mathcal{A} \subset \mathit{Adv}$ of discrete-progress adversaries of $\mathcal{S}^{\phi}_{T}$, the satisfaction relation $s \models_{\mathcal{A}} \phi$ is defined inductively as follows:*

$$
\begin{array}{lcl}
s \models_{\mathcal{A}} \mathtt{true} & & \text{for all } s \in S \\
s \models_{\mathcal{A}} a & \Leftrightarrow & a \in L(s) \\
s \models_{\mathcal{A}} \zeta & \Leftrightarrow & s \in \zeta \\
s \models_{\mathcal{A}} \phi_1 \wedge \phi_2 & \Leftrightarrow & s \models_{\mathcal{A}} \phi_1 \text{ and } s \models_{\mathcal{A}} \phi_2 \\
s \models_{\mathcal{A}} \neg\phi & \Leftrightarrow & s \not\models_{\mathcal{A}} \phi \\
s \models_{\mathcal{A}} z.\phi & \Leftrightarrow & s[z := 0] \models_{\mathcal{A}} \phi \\
s \models_{\mathcal{A}} [\phi_1 \,\exists\, \mathcal{U}\, \phi_2]_{\sqsupseteq\lambda} & \Leftrightarrow & \mathit{Prob}(\{\omega \in \mathit{Path}^{A}_{\mathit{ful}}(s) \mid \omega \models_{\mathcal{A}} \phi_1 \,\mathcal{U}\, \phi_2\}) \sqsupseteq \lambda \\
 & & \text{for some } A \in \mathcal{A} \\
s \models_{\mathcal{A}} [\phi_1 \,\forall\, \mathcal{U}\, \phi_2]_{\sqsupseteq\lambda} & \Leftrightarrow & \mathit{Prob}(\{\omega \in \mathit{Path}^{A}_{\mathit{ful}}(s) \mid \omega \models_{\mathcal{A}} \phi_1 \,\mathcal{U}\, \phi_2\}) \sqsupseteq \lambda \\
 & & \text{for all } A \in \mathcal{A} \\
\omega \models_{\mathcal{A}} \phi_1 \,\mathcal{U}\, \phi_2 & \Leftrightarrow & \text{there exists a position } (i, \delta) \text{ of } \omega \text{ such that} \\
 & & \omega(i) + \delta \ \models_{\mathcal{A}} \ \phi_2, \text{ and} \\
 & & \text{for all positions } (j, \delta') \text{ of } \omega \text{ such that } (j, \delta') \ll (i, \delta), \\
 & & \omega(j) + \delta' \ \models_{\mathcal{A}} \ \phi_1 \vee \phi_2 \,.
\end{array}
$$

5.3 Region equivalence

We now recall the definition of clock equivalence [AD94], which is an equivalence relation on the set of clock valuations used as a basis of region equivalence. Consider the probabilistic timed automaton T (which features the clock set $\mathcal{X} = \{x_1, ..., x_n\}$), and a PTCTL formula ϕ (which features clocks both from $\mathcal{X}$ and the formula clock set $\mathcal{Z} = \{z_1, ..., z_m\}$). Let $c_{\max}(T, \phi)$ be the maximal constant that any system clock $x \in \mathcal{X} \cup \mathcal{Z}$ is compared to in any of the invariant or enabling conditions of T, or in a zone subformula of ϕ. For any $t \in \mathbb{R}_{\geq 0}$, we use $\lfloor t \rfloor$ to denote the integral part of t, and $fr(t)$ to denote its fractional part; that is, $t = \lfloor t \rfloor + fr(t)$.

Definition 5.3.1 (Agreement on integer parts) *Let $t, t' \in \mathbb{R}_{\geq 0}$ be non-negative real values. Then t and t'* agree on integer parts *if:*

1. *$\lfloor t \rfloor = \lfloor t' \rfloor$, and*
2. *$fr(t) = 0$ if and only if $fr(t') = 0$.*

Definition 5.3.2 (Clock equivalence cf. [AD94]) *Consider the probabilistic timed automaton T and the PTCTL formula ϕ, and let $c = c_{\max}(T, \phi)$. For the valuations $\mathbf{a}, \mathbf{b} \in \mathbb{R}^{n+m}_{\geq 0}$, we have $\mathbf{a} \equiv^c \mathbf{b}$ if and only if the following conditions are satisfied:*

1. *for each $x_i \in \mathcal{X} \cup \mathcal{Z}$, either $\mathbf{a}_i$ and $\mathbf{b}_i$ agree on fractional parts, or $\mathbf{a}_i > c$ and $\mathbf{b}_i > c$; and*
2. *for each $x_i, x_j \in \mathcal{X} \cup \mathcal{Z}$, either $\mathbf{a}_i - \mathbf{a}_j$ and $\mathbf{b}_i - \mathbf{b}_j$ agree on fractional parts, or both $\mathbf{a}_i - \mathbf{a}_j > c$ and $\mathbf{b}_i - \mathbf{b}_j > c$.*

We use $[\mathbf{a}]$ to denote the equivalence class of $\equiv^c$ to which $\mathbf{a}$ belongs. We also write $\mathsf{Clock}^{\phi}_{T}(\mathcal{X} \cup \mathcal{Z})$ for the set of clock equivalence classes of the probabilistic timed automaton T and the PTCTL formula ϕ. Sometimes, when the context is clear, we simply write $\mathsf{Clock}(\mathcal{X} \cup \mathcal{Z})$.

Clock equivalence define a subclass of zones in the set $\mathsf{Zone}(\mathcal{X} \cup \mathcal{Z})$, the identification of which follows directly from Definition 5.3.2. Consider the case in which the probabilistic timed automaton in question has two clocks x_1 and x_2, and the set of formula clocks is empty. Clock equivalence subdivides the two-dimensional space of valuations into a grid, where, for each clock, a grid line corresponds to an integer value of the clock which is less than or equal to the maximal constant that the clock is compared to, either in the conditions of T or the zones featured in ϕ. The space is further subdivided by 45° diagonal lines passing through the intersection of each line with its axis. A *region* is a zone which consists of all of the valuations in a particular region equivalence class. Therefore, it takes the form of either:

- a corner point of the grid (for example, the point given by $\alpha_1 = (x_1 = 1 \wedge x_2 = 2)$);

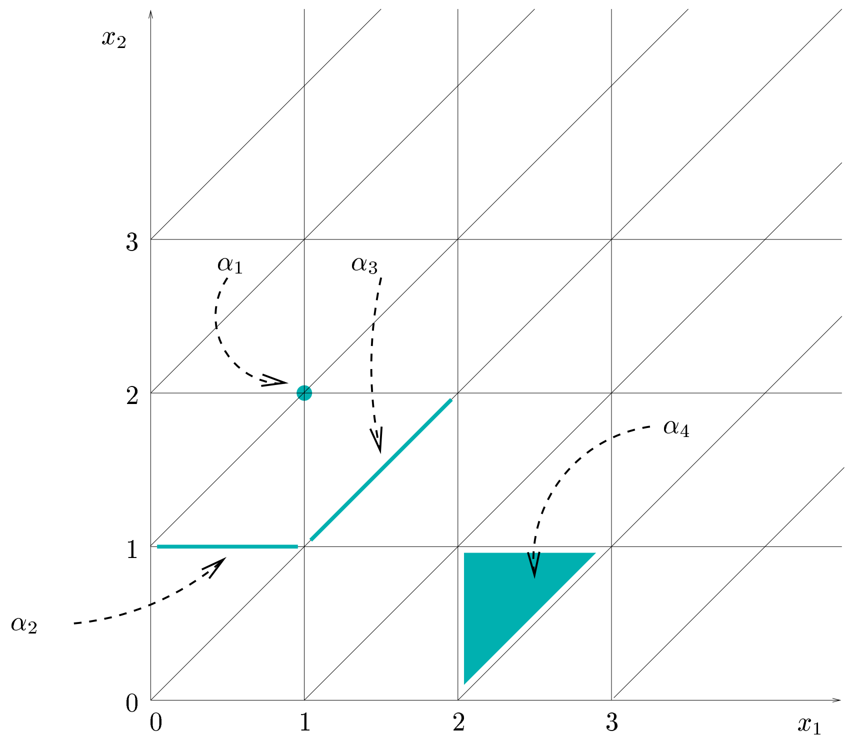


Figure 5.1: Clock equivalence for $c_{\max}(T, \phi) = 3$.

- a horizontal, vertical or diagonal open line segment bounded by (and excluding) corner points (such as a those given by $\alpha_2 = (x_1 > 0 \wedge x_1 < 1 \wedge x_2 = 1)$ or $\alpha_3 = (x_2 > 1 \wedge x_2 < 2 \wedge x_1 = x_2)$);
- an open triangular zone bounded by (and excluding) corner points, and horizontal, vertical and diagonal segments (such as the triangle given by $\alpha_4 = (x_1 > 2 \wedge x_2 < 1 \wedge x_1 < x_2 + 2)$).

Figure 5.1 illustrates the manner in which the continuous state space of valuations of the clocks x_1, x_2 is partitioned according to clock equivalence for the case in which $c_{\max}(T, \phi) = 3$. The clock equivalence classes corresponding to the examples $\alpha_1, ..., \alpha_4$ are also shown.

Certain characteristics of clock equivalence classes are summarised in the following lemma.

Lemma 5.3.3 *Consider the probabilistic timed automaton T and the PTCTL formula ϕ, and let $c = c_{\max}(T, \phi)$. Let $\mathbf{a}, \mathbf{b} \in \mathbb{R}_{\geq 0}^{n+m}$ such that $\mathbf{a} \equiv^c \mathbf{b}$. Then the following conditions hold:*

1. $\mathbf{a}[X := 0] \equiv^c \mathbf{b}[X := 0]$ *for each* $X \subseteq \mathcal{X} \cup \mathcal{Z}$,
2. *for all* $\delta \in \mathbb{R}_{\geq 0}$, *there exists* $\delta' \in \mathbb{R}_{\geq 0}$ *such that* $\mathbf{a} + \delta \equiv^c \mathbf{b} + \delta'$,

3. $\mathbf{a} \in \zeta$ *if and only if* $\mathbf{b} \in \zeta$ *for a zone* $\zeta \in \mathsf{Zone}(T, \phi)$*.*

Proof. The proof follows from the definition of $\equiv^c$. □

The first part of Lemma 5.3.3 states that, for clock equivalent valuations $\mathbf{a}, \mathbf{b} \in \mathbb{R}_{\geq 0}^{n+m}$, the valuation obtained after resetting the clocks in X to 0 in $\mathbf{a}$ is clock equivalent to the valuation obtained after performing the same operation on $\mathbf{b}$; that is, $\mathbf{a}[X := 0] \equiv^c \mathbf{b}[X := 0]$. Therefore, the set of valuations obtained after performing such a clock reset operation on all of the valuations in a region is itself a region. Hence, for a clock equivalence class $\alpha \in \mathsf{Clock}(\mathcal{X} \cup \mathcal{Z})$, we can write $\alpha[X := 0]$ for the set $\{\mathbf{b} \in \mathbb{R}_{\geq 0}^{n+m} \mid \mathbf{b} = \mathbf{a}[X := 0], \mathbf{a} \in \alpha\}$ (which, similarly, is also a clock equivalence class).

The second part of the lemma states that the same set of clock equivalence classes can be reached from any clock equivalent valuations by letting time elapse (which corresponds to the addition of a time duration such as δ or δ' to all of the variables in the valuations).

The third part of Lemma 5.3.3 states that, if a state in a clock equivalence class is a member of a zone used in the description of the probabilistic timed automaton T or the PTCTL formula ϕ, then all states in the class belong to the zone. This follows from the intuition that a zone of T or ϕ cannot describe anything "smaller" than a clock equivalence class; conversely, any zone of T or ϕ can be regarded as a union of clock equivalence classes, for some clock equivalence relation, and provided that the resulting set of valuations is convex. Note that, because of the presence of the maximal constant $c_{\max}(T, \phi)$ used in the definition of clock equivalence, it is *not* the case that an arbitrary zone from the set $\mathsf{Zone}(\mathcal{X} \cup \mathcal{Z})$ is a union of clock equivalence classes. For example, in Figure 5.1, we have $c_{\max}(T, \phi) = 3$, and therefore the zone denoted by the atomic formula $x > 4$ is *not* a union of clock equivalence classes given the maximal constant $c_{\max}(T, \phi)$.

Clock equivalence on valuations can be extended to an equivalence relation on states by requiring that their locations agree and their valuations are clock equivalent. We call this relation *region equivalence.*

Definition 5.3.4 (Region equivalence) *Let T be a probabilistic timed automaton, and ϕ be a PTCTL formula. Two states $(v, \mathbf{a}), (v', \mathbf{b}) \in S$ of the concurrent probabilistic system $\mathcal{S}_T^\phi$ are* region equivalent*, denoted by $(v, \mathbf{a}) \equiv^c_{region} (v', \mathbf{b})$, if and only if $v = v'$ and $\mathbf{a} \equiv^c \mathbf{b}$.*

Region equivalence classes are called *regions*. We write $\mathsf{Region}_T^\phi(\mathcal{X} \cup \mathcal{Z})$ to be the set of all regions of the probabilistic timed automaton T and the PTCTL formula ϕ, or, when clear from the context, this set is simply written as $\mathsf{Region}(\mathcal{X} \cup \mathcal{Z})$. It is often convenient to denote a region as a pair (v, α), where $v \in V$ is a location of T, and α is a clock equivalence class; that is, any state $(v', \mathbf{a}) \in S$ is a member of the region equivalence class $(v, \mathbf{a})$, written $(v', \mathbf{a}) \in (v, \alpha)$, if and only if $v = v'$ and $\mathbf{a} \in \alpha$. Note that we also sometimes write regions in the form $(v, [\mathbf{a}])$.

5.4 The probabilistic region graph

The definition of region equivalence presented in the previous section is now used to define a finite-state concurrent probabilistic system representing a probabilistic timed automaton T with respect to the PTCTL formula ϕ. The states of this system correspond to region equivalence classes, and the transitions correspond to the transitions available between states of T that belong to such classes. The resulting concurrent probabilistic system is called the *probabilistic region graph*. In Section 5.4.1, we define the probabilistic region graph, and equip this system with a notion of progress in Section 5.4.2. A fundamental concept for expressing the relationship between paths of the probabilistic timed automaton and the probabilistic region graph is presented in Section 5.4.3, and is extended to adversaries in Section 5.4.4.

5.4.1 Definition of the probabilistic region graph

We now define a transition relation over the set of regions of the concurrent probabilistic system $\mathcal{S}_T^\phi$. This transition relation is derived from that of the probabilistic timed automaton T in an intuitive manner, with the PTCTL formula ϕ being largely unimportant for this step. In particular, our definition of transitions between region equivalence classes is inspired by the non-probabilistic precedent of [AD94, ACD93]. Recall that the region construction of Alur et al. results in a finite-state labelled transition system. In contrast, deriving a natural transition relation between region equivalence classes of a probabilistic timed automaton results in a finite *concurrent probabilistic system*. In the following, we assume that any probabilistic timed automata subject to the region graph construction have non-deadlocking invariants.

States

Naturally, the states of the probabilistic region graph are the regions defined with respect to the probabilistic timed automaton T and the PTCTL formula ϕ of interest. The set of initial states comprises all the regions for which the values of all clocks of T are set to 0, while the value of the formula clocks in $\mathcal{Z}$ may be arbitrary.

Time transitions

Firstly, we introduce the following notation to reason about the first clock equivalence class reached from a given clock equivalence class after some time has elapsed.

Definition 5.4.1 (Time successor clock equivalence class) *Consider the distinct clock equivalence classes $\alpha, \beta \in \mathsf{Clock}(\mathcal{X} \cup \mathcal{Z})$. The class β is said to be the* time successor *of α if and only if, for each $\mathbf{a} \in \alpha$, there exists $\delta \in \mathbb{R}_{\geq 0}$ such that $\mathbf{a} + \delta \in \beta$, and $\mathbf{a} + \delta' \in \alpha \cup \beta$ for all $\delta' \leq \delta$. We then denote the equivalence class β by $succ(\alpha)$.*

The time successor relation can be extended to regions in the following way. We say that $(v', \beta) \in \mathsf{Region}(\mathcal{X} \cup \mathcal{Z})$ *is the* time successor region *of* $(v, \alpha) \in \mathsf{Region}(\mathcal{X} \cup \mathcal{Z})$ *if* $v' = v$ *and* $\beta = succ(\alpha)$*.*

If there exists a time successor region $(v, succ(\alpha))$ of a given region (v, α) such that $succ(\alpha)$ does not violate the location's invariant condition, then there is a *time transition* from (v, α) to $(v, succ(\alpha))$. Such a time transition takes the form of a Dirac distribution $\mathcal{D}(v, succ(\alpha))$.

Discrete transitions

Secondly, there will also be a set of probabilistic, discrete transitions associated with a region (v, α). The distributions comprising these transitions are derived from those distributions in $prob(v)$; more precisely, if α satisfies the enabling condition of a distribution of $prob(v)$, then there exists a corresponding distribution available in (v, α). If more than one tuple $(v', \mathbf{0}, X)$ in the support of p results in the same target region (v', β) when applied to the region (v, α), then the probability of making a transition to (v', β), denoted by $\bar{p}(v', \beta)$, is the sum of the probabilities assigned to such tuples by the distribution p.

Labelling function

The labelling function of the probabilistic region graph reflects both the labelling function of T, and the satisfaction of the zones appearing as subformulae of ϕ by regions. Firstly, note that a zone $\zeta \in \mathsf{Zone}(T, \phi)$ refers to the clocks of a probabilistic timed automaton and the formula clocks; therefore, the satisfaction of ζ depends only on the clock equivalence class component of a region. We say that a clock equivalence class $\alpha \in \mathsf{Clock}(\mathcal{X} \cup \mathcal{Z})$ satisfies a zone $\zeta \in \mathsf{Zone}(T, \phi)$ if and only if all valuations contained within α are also contained within ζ; that is $\alpha \subseteq \zeta$ (given that we note that either $\alpha \subseteq \zeta$ or $\alpha \cap \zeta = \emptyset$, by Lemma 5.3.3 part 3). Intuitively, α satisfies ζ if and only if the substitution of all valuations contained within α for the appropriate variables featured in the atomic formulae of ζ means that the conjunction of the clock constraints of ζ resolves to `true`. Then, a region (v, α) satisfies ζ if and only if α satisfies ζ. Next, for every zone $\zeta \in \mathsf{Zone}(T, \phi)$ appearing as a subformula of ϕ, we add the atomic proposition a_ζ to the set of atomic propositions; that is, the new set of atomic propositions [AP] is defined as $[\mathrm{AP}] = \mathrm{AP} \cup \{a_\zeta \mid \zeta \in \mathsf{sub}(\phi) \cap \mathsf{Zone}(T, \phi)\}$. We label each region (v, α) with the atomic propositions in $\mathcal{L}(v)$ and the atomic proposition a_ζ, for each zone $\zeta \in \mathsf{sub}(\phi) \cap \mathsf{Zone}(T, \phi)$ such that $\alpha \subseteq \zeta$.

Formal definition of the probabilistic region graph

We are now able to define formally the probabilistic region graph.

Definition 5.4.2 (Probabilistic region graph) *The* probabilistic region graph *is the concurrent probabilistic system $[\mathcal{S}_T^\phi] = ([S], [\bar{S}], [L], [\Sigma], [Steps])$ defined in the following way.*

- *The state set is the set of regions; that is, $[S] = \mathsf{Region}_T^\phi(\mathcal{X} \cup \mathcal{Z})$.*
- *The initial state set $[\bar{S}] \subseteq [S]$ is the set of regions of the form $(\bar{v}, [\mathbf{0}_{|\mathcal{X}}])$.*
- *The labelling function $[L] : [S] \to 2^{[\mathrm{AP}]}$ is defined as:*

$$[L](v, \alpha) = \mathcal{L}(v) \cup \{a_\zeta \in [\mathrm{AP}] \mid \alpha \subseteq \zeta, \zeta \in \mathsf{sub}(\phi) \cap \mathsf{Zone}(\mathcal{X} \cup \mathcal{Z})\}\,.$$

- *The event set is $[\Sigma] = \{\theta_p \mid p \in prob(v), v \in V\} \cup \{\tau\}$.*
- *The transition relation $[Steps] : [S] \to ([\Sigma] \times \mu([S]))$ is characterised in the following way. For each region $(v, \alpha) \in [S]$, let $[Steps](v, \alpha) = [Time](v, \alpha) \cup [Disc](v, \alpha)$ be the smallest set of event-distribution pairs such that:*

  time transition *if there exists a time successor $succ(\alpha)$ of α such that $succ(\alpha) \subseteq inv(v)$, then $[Time](v, \alpha) = \{(\tau, \mathcal{D}(v, succ(\alpha)))\}$, otherwise $[Time](v, \alpha) = \emptyset$;*

  discrete transitions *there exists $(\theta, \bar{p}) \in [Disc](v, \alpha)$ if and only if there exists a distribution $p \in prob(v)$ and $\alpha \subseteq pre_v(p)$, and where $\bar{p}$ is such that, for any region $(v, \beta) \in [S]$:*

$$\bar{p}(v', \beta) = \sum_{\substack{X \subseteq \mathcal{X}\, \& \\ \beta = \alpha[X:=0]}} p(v', X)\,.$$

Noting that the probabilistic region graph $[\mathcal{S}_T^\phi]$ is a finite concurrent probabilistic system, we can define and reason about its paths in the usual way. We typically use π to denote paths of $[\mathcal{S}_T^\phi]$ to distinguish its paths from those of $\mathcal{S}_T^\phi$, which are denoted by ω. Often, for a time transition $(v, \alpha) \xrightarrow{\tau, \mathcal{D}(v, succ(\alpha))} (v, succ(\alpha))$, we write $(v, \alpha) \xrightarrow{\tau} (v, succ(\alpha))$.

5.4.2 Progress in the probabilistic region graph

It is important to have some notion of *divergence* of the paths of $[\mathcal{S}_T^\phi]$ if we wish our model checking result on $[\mathcal{S}_T^\phi]$ to reflect accurately the progress assumptions made on the concurrent probabilistic system $\mathcal{S}_T^\phi$ in Section 4.4. We now introduce discrete- and time-progress for probabilistic region graph paths, and establish their correspondence with discrete- and time-progress on the paths of the concurrent probabilistic system of T. Then it follows that the combination of the two concepts of progress can be used to define divergent adversaries on the probabilistic region graph.

Firstly, recall Definition 4.4.1, which expresses discrete progress on the paths of concurrent probabilistic systems representing the underlying semantics of probabilistic timed automata. Clearly, each discrete-progress path on the probabilistic region graph should also have infinitely many probabilistic discrete transitions, but, unlike the case of Definition 4.4.1, we do not require alternation between probabilistic transitions and time transitions. There are two reasons for this choice: firstly, time transitions of the probabilistic region graph correspond to the passage of $\delta > 0$ time units, whereas a concurrent probabilistic system representing the semantics of a probabilistic timed automaton can take time transitions of duration 0. Secondly, it may be the case that more than 1 time unit passes before the probabilistic timed automaton makes a discrete transition, and yet passing from one region to its time successor corresponds to the passage of less than 1 time unit.

The following definition of discrete progress in the probabilistic region graph essentially agrees with the notion of discrete progress in the concurrent probabilistic system of a probabilistic timed automaton, in that both require an infinite number of discrete transitions occurring on a path. Recall that τ is the event used to denote a time-transition in the probabilistic region graph.

Definition 5.4.3 (Discrete progress in the probabilistic region graph) *Let T be a probabilistic timed automaton and ϕ be a PTCTL formula. For the probabilistic region graph $[\mathcal{S}_T^\phi]$, the path*

$$\pi = (v_0, \alpha_0) \xrightarrow{\sigma_0, \bar{p}_0} (v_1, \alpha_1) \xrightarrow{\sigma_1, \bar{p}_1} (v_2, \alpha_2) \xrightarrow{\sigma_2, \bar{p}_2} \ldots ,$$

where $\pi \in Path_{ful}^{[\mathcal{S}_T^\phi]}$, exhibits discrete progress *if, for all $i \in \mathbb{N}$, there exists $j > i$ such that $\sigma_j \neq \tau$.*

Secondly, we address the notion of time progress along probabilistic region graph paths. The following notation is introduced to classify certain regions.

x_i-zero class For a clock $x_i \in \mathcal{X} \cup \mathcal{Z}$, the clock equivalence class α is said to be *x_i-zero* if and only if $\mathbf{a}_i = 0$ for each $\mathbf{a} \in \alpha$.

x_i-unbounded class For a clock $x_i \in \mathcal{X} \cup \mathcal{Z}$, the equivalence class α is said to be *x_i-unbounded* if and only if $\mathbf{a}_i > c_{\max}(T, \phi)$ for each $\mathbf{a} \in \alpha$.

The region (v, α) is x_i-zero (x_i-unbounded) if α is x_i-zero (x_i-unbounded).

The following definition is inspired by the non-probabilistic precedent of [BTY97, Tri98].

Definition 5.4.4 (Time progress in the probabilistic region graph) *Let T be a probabilistic timed automaton and ϕ be a PTCTL formula. Consider the path of the probabilistic region graph $[\mathcal{S}_T^\phi]$ denoted by*

$$\pi = (v_0, \alpha_0) \xrightarrow{\sigma_0, \bar{p}_0} (v_1, \alpha_1) \xrightarrow{\sigma_1, \bar{p}_1} (v_2, \alpha_2) \xrightarrow{\sigma_2, \bar{p}_2} \ldots ,$$

where $\pi \in Path_{ful}^{[\mathcal{S}_T^\phi]}$.

Possibly progressive *The path* π *is* possibly progressive *if and only if, for each clock* $x \in \mathcal{X} \cup \mathcal{Z}$*, either:*

1. *for every* $i \in \mathbb{N}$*, there exists* $j \geq i$ *such that* α_j *is an* x*-zero class, or*
2. *there exists* $i \in \mathbb{N}$ *such that, for all* $j \geq i$*,* α_j *is an* x*-unbounded class.*

Zero *The path* π *is* zero *if and only if there exists a clock* $x \in \mathcal{X}$ *and* $i \in \mathbb{N}$ *such that, for all* $j > i$*,* α_j *is* x*-zero.*

Time progress *The path* π *exhibits* time progress *if and only if it is possibly progressive and not zero.*

Definition 5.4.5 (Divergent probabilistic region graph paths) *Let* T *be a probabilistic timed automaton and* ϕ *be a PTCTL formula. For the probabilistic region graph* $[\mathcal{S}_T^\phi]$*, a path* $\pi \in Path_{ful}^{[\mathcal{S}_T^\phi]}$ *is* divergent *if and only if it exhibits discrete progress and time progress.*

5.4.3 Path inscription

In order to establish the correspondence between paths of $[\mathcal{S}_T^\phi]$ and those of $\mathcal{S}_T^\phi$, we adopt the notion of inscription of paths of [Tri98] to our context. Intuitively, a path ω of $\mathcal{S}_T^\phi$ is inscribed within a path π of $[\mathcal{S}_T^\phi]$ if ω can be regarded as being in the set of probabilistic timed automaton paths which the probabilistic region graph path π represents. To facilitate the definition, we alter slightly the indexing of the states and events along π. That is, a path π of $[\mathcal{S}_T^\phi]$ will be written as:

$$\pi = (v_0, \alpha_{0,0}) \xrightarrow{\tau} \ldots \xrightarrow{\tau} (v_0, \alpha_{0,l_0}) \xrightarrow{\theta_{p_0}, \bar{p}_0} (v_1, \alpha_{1,0}) \xrightarrow{\tau} \ldots \xrightarrow{\tau} (v_1, \alpha_{1,l_1}) \xrightarrow{\theta_{p_1}, \bar{p}_1} \ldots$$

where $l_i \in \mathbb{N}$ for each $i \in \mathbb{N}$. Therefore, the path π is regarded as an infinite concatenation of segments of the form $(v_i, \alpha_{i,0}) \xrightarrow{\tau} \ldots \xrightarrow{\tau} (v_i, \alpha_{i,l_i}) \xrightarrow{\theta_{p_i}, \bar{p}_i}$, which represent a sequence of time transitions, followed by a probabilistic transition, of $[\mathcal{S}_T^\phi]$. Throughout this section, we assume that the probabilistic timed automaton T and the PTCTL formula ϕ are fixed. **Henceforth, to ease notation, we omit the ϕ superscript and T subscript from the concurrent probabilistic system $\mathcal{S}_T^\phi$ and the probabilistic region graph $[\mathcal{S}_T^\phi]$, to result in $\mathcal{S}$ and $[\mathcal{S}]$ respectively.** Also for convenience, we often abuse notation by subscripting valuations with natural numbers; in this section, this is done to distinguish different valuations, and *not* to refer to the value assigned by the valuation to the variable corresponding to the subscript.

Definition 5.4.6 (Probabilistic region graph inscription) *Let T be a probabilistic timed automaton and ϕ be a PTCTL formula. Consider the path of $\mathcal{S}$ denoted by*

$$\omega = (v_0, \mathbf{a}_0) \xrightarrow{\delta_0}\xrightarrow{\theta_{p_0}, \tilde{p}_0} (v_1, \mathbf{a}_1) \xrightarrow{\delta_1}\xrightarrow{\theta_{p_1}, \tilde{p}_1} \dots ,$$

and let:

$$\pi = (v_0, \alpha_{0,0}) \xrightarrow{\tau} \dots \xrightarrow{\tau} (v_0, \alpha_{0,l_0}) \xrightarrow{\theta_{p_0}, \tilde{p}_0} (v_1, \alpha_{1,0}) \xrightarrow{\tau} \dots \xrightarrow{\tau} (v_1, \alpha_{1,l_1}) \xrightarrow{\theta_{p_1}, \tilde{p}_1} \dots$$

be a path of $[\mathcal{S}]$. Then ω is inscribed *in π if $\alpha_{i,0} = [\mathbf{a}_i]$ for all $i \in \mathbb{N}$.*

We now show how a path π_ω of $[\mathcal{S}]$ can be constructed from a path ω of $\mathcal{S}$, and, conversely, how a path ω_π of $\mathcal{S}$ can be constructed from a path π of $[\mathcal{S}]$, such that the path of $[\mathcal{S}]$ inscribes that of $\mathcal{S}$. First, we deal with the case of requiring discrete progress, and then turn our attention to time progress.

Lemma 5.4.7 *Let T be a probabilistic timed automaton, ϕ be a PTCTL formula, $\mathcal{S}$ be the concurrent probabilistic system of T and ϕ, and $[\mathcal{S}]$ be the probabilistic region graph. Then there exists a discrete-progress path $\omega \in Path^{\mathcal{S}}_{ful}$ of $\mathcal{S}$, if and only if there exists a discrete-progress path $\pi \in Path^{[\mathcal{S}]}_{ful}$ of $[\mathcal{S}]$, such that ω is inscribed in π.*

Proof. Our proof is based on a similar result in [ACD93]. Let $c = c_{\max}(T, \phi)$. Consider the following property, which will henceforth be referred to as the *sequence property.* Take a particular state $(v, \mathbf{a}) \in S$ of $\mathcal{S}$, and consider the sequence of clock equivalence classes, $[\mathbf{a}], [\mathbf{a} + \delta_1], [\mathbf{a} + \delta_2], ..., [\mathbf{a} + \delta_k]$ for a finite $k \geq 1$, where $succ([\mathbf{a}]) = [\mathbf{a} + \delta_1]$, $\delta_i \in \mathbb{R}_{\geq 0}$ and $[\mathbf{a} + \delta_i] \subseteq inv(v)$ for each $1 \leq i \leq k$, and, if $k > 1$, we also have $succ([\mathbf{a} + \delta_i]) = [\mathbf{a} + \delta_{i+1}]$ for each $1 \leq i < k$. Then, for the duration $\delta \in \mathbb{R}_{\geq 0}$, where $\delta_1 \leq \delta \leq \delta_k$, we know that $\mathbf{a} + \delta \equiv^c \mathbf{a} + \delta_j$ for some $1 \leq j \leq k$. We can then write $\mathbf{a} + \delta \in [\mathbf{a} + \delta_j]$.

The two directions of the lemma are treated separately. First, we construct a discrete-progress path π_ω of $[\mathcal{S}]$ from the discrete-progress path ω of $\mathcal{S}$ such that ω is inscribed in π_ω; then we consider the opposite direction, and construct ω_π of $\mathcal{S}$ from π of $[\mathcal{S}]$ such that ω_π is inscribed in π.

Construction of π_ω from ω. Consider the path $\omega \in Path^{\mathcal{S}}_{ful}$ of $\mathcal{S}$ which exhibits discrete progress, and which is of the form

$$\omega = (v_0, \mathbf{a}_0) \xrightarrow{\delta_0}\xrightarrow{\tilde{p}_0} (v_1, \mathbf{a}_1) \xrightarrow{\delta_1}\xrightarrow{\tilde{p}_1} (v_2, \mathbf{a}_2) \xrightarrow{\delta_2}\xrightarrow{\tilde{p}_2} \cdots$$

Take a particular $i \in \mathbb{N}$. From the state $(v_i, \mathbf{a}_i)$, and letting δ_i time units elapse, we may cross into a number of new equivalence classes before making the next discrete transition. We let $l_i \in \mathbb{N}$ be this number. Then, by the sequence property and

Lemma 5.3.3 part 2, we can construct the finite path π_i of the probabilistic region graph, such that:

$$\pi_i = (v_i, \alpha_{i,0}) \xrightarrow{\tau} (v_i, \alpha_{i,1}) \xrightarrow{\tau} \dots \xrightarrow{\tau} (v_i, \alpha_{i,l_i})$$

where $\alpha_{i,0} = [\mathbf{a}_i]$, and $\alpha_{i,j} = [\mathbf{a}_i + \delta_{i,j}]$ for some $\delta_{i,j} < \delta_i$. (Observe that it may be the case that π_i comprises the single region $(v_i, \alpha_{i,0})$; that is, the passage of δ_i time units means that the system stays in the same region equivalence class.) Now note that $(v_i, \mathbf{a}_i + \delta_i) \in (v_i, \alpha_{i,l_i})$. As all of the clock valuations in α_{i,l_i} will enable the same probability distributions of $prob(v_i)$, then the same distributions are enabled in $(v_i, \mathbf{a}_i + \delta_i)$ and (v_i, α_{i,l_i}). Therefore, the *same* distribution $p_i \in prob(v_i)$ as that chosen in the $(i+1)$th transition of ω, denoted by $(v_i, \mathbf{a}_i) \xrightarrow{\delta_i} \xrightarrow{\tilde{p}_i} (v_{i+1}, \mathbf{a}_{i+1})$, is chosen as a basis for making the next transition from (v_i, α_{i,l_i}). Then the event-distribution pair for making the next transition will be $(\theta_{p_i}, \bar{p}_i)$. It follows that, for whichever tuple $(v_{i+1}, X) \in \mathsf{support}(p_i)$ used to make the discrete transition in $(v_i, \mathbf{a}_i) \xrightarrow{\delta_i} \xrightarrow{\tilde{p}_i} (v_{i+1}, \mathbf{a}_{i+1})$, then this same tuple can be used to derive the transition $(v_i, \alpha_{i,l_i}) \xrightarrow{\theta_p, \bar{p}_i} (v_{i+1}, \alpha_{i+1,0})$ of the probabilistic region graph. This follows from the fact that, by part 1 of Lemma 5.3.3, we have $(\mathbf{a}_i + \delta_i)[X := 0] \in \alpha_{i,l_i}[X := 0]$. Then we let:

$$\pi_i' = (v_i, \alpha_{i,0}) \xrightarrow{\tau} (v_i, \alpha_{i,1}) \xrightarrow{\tau} \dots \xrightarrow{\tau} (v_i, \alpha_{i,l_i}) \xrightarrow{\theta_{p_i}, \bar{p}_i} (v_{i+1}, \alpha_{i+1,0})$$

where $\alpha_{i+1,0} = \alpha_{i,l_i}[X := 0]$. Let $\pi_\omega = \pi_0' \pi_1' \cdots$ be the concatenation of all such segments. Therefore, we have constructed the path π_ω from ω such that π_ω inscribes ω. As each path segment π_i' for all $i \in \mathbb{N}$ includes one discrete transition, the path π_ω exhibits discrete progress.

Construction of ω_π from π. The construction described in the previous paragraph also works in the opposite direction. Let

$$\pi = (v_0, \alpha_{0,0}) \xrightarrow{\tau} \dots \xrightarrow{\tau} (v_0, \alpha_{0,l_0}) \xrightarrow{\theta_{p_0}, \bar{p}_0} (v_1, \alpha_{1,0}) \xrightarrow{\tau} \dots \xrightarrow{\tau} (v_1, \alpha_{1,l_1}) \xrightarrow{\theta_{p_1}, \bar{p}_1} \dots$$

be a path $\pi \in Path_{ful}^{[\mathcal{S}]}$ of $[\mathcal{S}]$ which exhibits discrete progress. We proceed to construct the path ω_π of $\mathcal{S}$ inductively by progressing along the length of the path π. For the base case of length 0, let $(v_0, \mathbf{a}_0)$ be a state such that $\mathbf{a}_0 \in \alpha_{0,0}$; then $(v_0, \mathbf{a}_0)$ is the first state of the path ω_π.

For the inductive step, we have two cases for each $i \in \mathbb{N}$, depending on whether $l_i \geq 1$ or $l_i = 0$.

Case: $l_i \geq 1$. Let

$$\pi^{frag} = (v_i, \alpha_{i,0}) \xrightarrow{\tau} (v_i, \alpha_{i,1}) \xrightarrow{\tau} \dots \xrightarrow{\tau} (v_i, \alpha_{i,l_i})$$

be the path fragment of π immediately following the ith discrete transition. Note that all of its transitions of π^{frag} correspond to time transitions. Now suppose that the transition following the path fragment π^{frag} is $(v_i, \alpha_{i,l_i}) \xrightarrow{\theta_{p_i}, \bar{p}_i} (v_{i+1}, \alpha_{i+1,0})$, where $\bar{p}_i$ is derived from the distribution p_i of T, and where $p_i \in \mathit{prob}(v_i)$. Assume by induction that we have a partially constructed, finite path

$$\omega_\pi^{\mathit{fin}} = (v_0, \mathbf{a}_0) \xrightarrow{\delta_0} \xrightarrow{\tilde{p}_0} \dots \xrightarrow{\delta_{i-1}} \xrightarrow{\tilde{p}_{i-1}} (v_i, \mathbf{a}_i)$$

of $\mathcal{S}$ such that $|\omega_\pi^{\mathit{fin}}| = 2i$, and $\mathbf{a}_i \in \alpha_i$. Then it follows that $\omega_\pi^{\mathit{fin}}$ can be extended by the transition $(v_i, \mathbf{a}_i) \xrightarrow{\delta_i} \xrightarrow{\tilde{p}_i}$, where $\delta_i \in \mathbb{R}_{\geq 0}$ is such that $\mathbf{a}_i + \delta_i \in \alpha_{i,l_i} \subseteq \mathit{pre}_v(p)$, and $\tilde{p}_i$ is derived from p_i in the manner of Definition 4.3.2. Using a symmetric argument to that used in the construction of π_ω, we can find a state $(v_{i+1}, \mathbf{a}_{i+1}) \in \mathsf{support}(\tilde{p}_i)$ which is such that $(v_{i+1}, \mathbf{a}_{i+1}) \in (v_{i+1}, \alpha_{i+1,0})$.

Case: $l_i = 0$ In this case, the path of $[\mathcal{S}]$ performs two consecutive probabilistic, discrete transitions. Then we can simply let $\delta_i = 0$, and derive $\tilde{p}_i$ from the transition $(v_i, \alpha_{i,0}) \xrightarrow{\theta_{p_i}, \bar{p}_i} (v_{i+1}, \alpha_{i+1,0})$ in the same way as given in the previous case.

Repeating this for all $i \geq 0$ gives us a method for constructing the discrete-progress path ω_π of $\mathcal{S}$ from the discrete-progress path π of $[\mathcal{S}]$. □

Note that, for every state $\bar{s}$ in the initial state set $\bar{S}$ of $\mathcal{S}$, there exists a state $[\bar{s}]$ in the initial state set $[\bar{S}]$ of $[\mathcal{S}]$ such that $\bar{s} \in [\bar{s}]$. Similarly, for every $[\bar{s}] \in [\bar{S}]$, there exists $\bar{s} \in \bar{S}$ such that $\bar{s} \in [\bar{s}]$. This is because $\bar{s}$ must be of the form $\mathbf{0}_{|\mathcal{X}}$, and $[\bar{s}]$ must be of the form $[\mathbf{0}_{|\mathcal{X}}]$. Hence, if the path ω (respectively, π) starts from an initial state of $\mathcal{S}$ (respectively, $[\mathcal{S}]$), then we can construct a path π (respectively, ω) such that ω is inscribed in π.

The fact that the construction process in the proof of Lemma 5.4.7 also permits time-progress paths to be constructed, in both directions, is formalised in the following lemma.

Lemma 5.4.8 *Let T be a probabilistic timed automaton, and ϕ be a PTCTL formula. For the concurrent probabilistic system $\mathcal{S}$ of T and ϕ, and the probabilistic region graph $[\mathcal{S}]$, there exists a divergent path $\omega \in \mathit{Path}_{\mathit{ful}}^{\mathcal{S}}$ of $\mathcal{S}$ if and only if there exists a divergent path $\pi \in \mathit{Path}_{\mathit{ful}}^{[\mathcal{S}]}$ of $[\mathcal{S}]$ such that ω is inscribed in π.*

Proof. First consider the claim that every path $\omega \in \mathit{Path}_{\mathit{ful}}^{\mathcal{S}}$ of $\mathcal{S}$ exhibiting time progress will be inscribed within a path $\pi_\omega \in \mathit{Path}_{\mathit{ful}}^{[\mathcal{S}]}$ of $[\mathcal{S}]$ which also exhibits time progress, where π_ω is constructed according to the method of the proof of Lemma 5.4.7. Suppose by contradiction that π_ω is a zero path, then by Definition 5.4.4 there exists $i \in \mathbb{N}$ and $x \in \mathcal{X} \cup \mathcal{Z}$ such that $\pi_\omega(j)$ is x-zero for all $j > i$. By the construction of π_ω in Lemma 5.4.7, this corresponds to the situation in which the value of the clock x does not advance from some point on in the path ω, and therefore time also cannot advance. However, this contradicts the fact that ω is divergent, and hence π_ω is not zero.

Secondly, for any clock $x \in \mathcal{X} \cup \mathcal{Z}$, if there exists $i \in \mathbb{N}$ such that there does not exist an x-zero region after the vertex $\pi_\omega(i)$, then the clock x is not reset after this point in the path. Therefore, since the total time elapsed in ω must exceed any bound, the value of the clock x must also exceed any bound. Therefore, there must exist a $j \geq i$ for which all regions appearing along the path after $\pi_\omega(j)$ are x-unbounded. We conclude that π_ω must be possibly progressive, and therefore π_ω is also divergent.

Conversely, we can show that, if the probabilistic region graph path $\pi \in Path_{ful}^{[\mathcal{S}]}$ exhibits time progress, then the path $\omega_\pi \in Path_{ful}^{\mathcal{S}}$ that is constructed from π according to the proof of Lemma 5.4.7 can be chosen to exhibit time progress. We sketch the ideas underlying this result. Firstly, as π is not an zero path, it follows that it is possible to select the states and transitions along ω_π such that a positive amount of time can elapse infinitely often. Secondly, as π is possibly progressive, it follows that the value of each clock is either reset infinitely often or is not bounded from above from some point along the path onwards. Therefore, the durations of time transitions are not forced to converge to 0 to prevent a clock from exceeding an upper bound. Hence, we can construct ω_π such that there exists some constant $\varepsilon > 0$ such that every time transition in ω_π is bounded from below by ε; then it must be the case that the duration of the path in the limit is unbounded, and therefore ω_π exhibits time progress. $\square$

Observe another pleasing correspondence between the paths of the probabilistic region graph and those of the concurrent probabilistic system of T and ϕ: for the paths $\omega \in Path_{ful}^{\mathcal{S}}$ of $\mathcal{S}$ and $\pi \in Path_{ful}^{[\mathcal{S}]}$ of $[\mathcal{S}]$, such that ω is inscribed in π, the probabilistic distributions along π will assign the same probability to the region (v', β) reached via the subsequent transition as that assigned by the corresponding distribution along ω to the states which are contained within (v', β). To formalise this, we present the following lemma.

Lemma 5.4.9 *Let T be a probabilistic timed automaton, and ϕ be a PTCTL formula. Consider the regions $(v, \alpha), (v', \beta) \in [S]$ of the probabilistic region graph $[\mathcal{S}]$ and the states $(v, \mathbf{a}), (v', \mathbf{b}) \in S$ of the concurrent probabilistic system $\mathcal{S}$ such that $(v, \mathbf{a}) \in (v, \alpha)$ and $(v', \mathbf{b}) \in (v', \beta)$. Then there will exist $(\theta_p, \tilde{p}) \in [Steps](v, \mathbf{a})$ if and only if there exists $(\theta_p, \bar{p}) \in Steps(v, \alpha)$ such that*

1. *we have $(v', \mathbf{b}) \in \mathsf{support}(\tilde{p})$ if and only if $(v', \beta) \in \mathsf{support}(\bar{p})$, and*
2. *$\bar{p}(v', \beta) = \tilde{p}(v', \mathbf{b})$.*

Proof. We sketch the proof of this lemma. First consider the fact that the existence of $(\theta_p, \_) \in Steps(v, \mathbf{a})$ implies the existence of $(\theta_p, \_) \in Steps(v, \alpha)$ (our concern is for the events, and therefore we temporarily substitute the place-holder symbol _ for the relevant distribution). As $(\theta_p, \_) \in \mathit{Steps}(v, \mathbf{a})$ implies that $\mathbf{a} \in \mathit{pre}_v(p)$ for $p \in \mathit{prob}(v)$, and part 3 of Lemma 5.3.3 implies that $\alpha \subseteq \mathit{pre}_v(p)$, then, by construction of the probabilistic region graph (Definition 5.4.2), there exists $(\theta_p, \_) \in \mathit{Steps}(v, \alpha)$.

Conversely, if $(\theta_p, \_) \in \mathit{Steps}(v, \alpha)$, then $\alpha \subseteq \mathit{pre}_v(p)$, and as $\mathbf{a} \in \alpha$, it must be the case that $\mathbf{a} \in \mathit{pre}_v(p)$; therefore, $(\theta_p, \_) \in \mathit{Steps}(v, \alpha)$ implies that $(\theta_p, \_) \in \mathit{Steps}(v, \mathbf{a})$.

We now consider each of the parts of the remainder of the lemma in turn.

Part 1. We focus attention to the way in which the distributions $\tilde{p}$ and $\bar{p}$ are derived from the distribution $p \in \mathit{prob}(v)$. Part 1 of the lemma follows from the arguments used in the proof of Lemma 5.4.7, which shows that, for a transition $(v, \mathbf{a}) \xrightarrow{\theta_p, \tilde{p}} (v', \mathbf{b})$ of the concurrent probabilistic system $\mathcal{S}$, we can find a transition $(v, \alpha) \xrightarrow{\theta_p, \hat{p}} (v', \beta)$ of the probabilistic region graph $[\mathcal{S}]$. The opposite direction of this is also shown in the proof of Lemma 5.4.7. Then we have $(v', \mathbf{b}) \in \mathsf{support}(\tilde{p})$ if and only if $(v', \beta) \in \mathsf{support}(\bar{p})$.

Part 2. Observe that any two distinct tuples $(v', X_1), (v', X_2) \in \mathsf{support}(p)$ can result in the same state $(v', \mathbf{b})$ after resetting the appropriate clocks in $(v, \mathbf{a})$ only if, for any $x_i \in (X_1 \setminus X_2) \cup (X_2 \setminus X_1)$, it is the case that $\mathbf{a}_i = 0$. That is, for the resetting of clocks in X_1 and X_2 to result in the same state, then the values of the clocks in X_1 but not in X_2, and in X_2 but not in X_1, must be 0 before the transition is taken. Then the information about which clocks are 0 before the transition is encoded in the region (v, α) which contains $(v, \mathbf{a})$. That is, all valuations $\mathbf{a}' \in \alpha$ are such that $\mathbf{a}'_i = 0$ for all $x_i \in (X_1 \setminus X_2) \cup (X_2 \setminus X_1)$. Therefore, the application of either of the two tuples $(v', X_1), (v', X_2)$ to the region (v, α) must result in the same region (v', β). Also, (v', β) must be such that $(v', \mathbf{b}) \in (v', \beta)$. Applying this intuition to sets of tuples, rather than just pairs, gives us that $\bar{p}(v', \beta) = \tilde{p}(v', \mathbf{b})$. Similar intuition can be applied in the reverse direction: that is, for any set of tuples resulting in the same region being reached from (v, α), then this set of tuples must also result in the same states being reached from $(v, \mathbf{a})$. □

5.4.4 Adversaries of the probabilistic region graph

We define *adversaries of the probabilistic region graph* in the standard manner for concurrent probabilistic systems. Note that an adversary of $[\mathcal{S}]$ corresponds to an infinite number of adversaries on the concurrent probabilistic system $\mathcal{S}$. This is because the duration δ in the choice of a transition $\xrightarrow{\delta}$ of $\mathcal{S}$ allows positive reals to be chosen, whereas the passage of time from the region (v, α) of $[\mathcal{S}]$ is represented by the single distribution $\mathcal{D}(v, \mathit{succ}(\alpha))$. We typically denote adversaries of $[\mathcal{S}]$ by B, with sets of such adversaries being denoted by $\mathcal{B}$, and the set of all adversaries denoted by $[\mathit{Adv}]$. From Lemma 5.4.7, we have the property that, for every path $\omega \in \mathit{Path}_{\mathit{ful}}^{\mathcal{S}}$ of $\mathcal{S}$, we can construct a path $\pi \in Path_{ful}^{[\mathcal{S}]}$ of $[\mathcal{S}]$ such that ω is inscribed in π. Conversely, for every path $\pi \in Path_{ful}^{[\mathcal{S}]}$ of $[\mathcal{S}]$, we can construct a path $\omega \in Path_{ful}^{\mathcal{S}}$ of $\mathcal{S}$ such that π inscribes ω. Therefore, given an adversary A of $\mathcal{S}$ which defines the set of paths $Path_{ful}^{\mathcal{S},A}$, we can construct an adversary B_A of $[\mathcal{S}]$ which defines the set of paths $Path_{ful}^{[\mathcal{S}],B_A}$, and, conversely, we can construct A_B of $\mathcal{S}$ from B of $[\mathcal{S}]$. Furthermore, the adversaries can be constructed so that every path in $Path_{ful}^{\mathcal{S},A}$ is inscribed in a path of $Path_{ful}^{[\mathcal{S}],B_A}$, and every path of $Path_{ful}^{[\mathcal{S}],B}$ inscribes a path of $Path_{ful}^{\mathcal{S},A_B}$. For notational convenience, as we denote adversaries of $\mathcal{S}$ by A, and those of $[\mathcal{S}]$ by B, we write $Path_{ful}^{A}$ for $Path_{ful}^{\mathcal{S},A}$ and $Path_{ful}^{B}$ for $Path_{ful}^{[\mathcal{S}],B}$.

The definition of divergent adversaries on $[\mathcal{S}]$ is with respect to our notion of divergence of paths presented in Definition 5.4.5.

Definition 5.4.10 (Divergent adversaries on the probabilistic region graph) *Let T be a probabilistic timed automaton, and ϕ be a PTCTL formula. An adversary B of the probabilistic region graph $[\mathcal{S}]$ is* divergent *if and only if*

$$Prob^{B}(\{\pi \in Path_{ful}^{B} \mid \pi \text{ is divergent}\}) = 1.$$

Let $[Div]$ be the set of divergent adversaries of $[\mathcal{S}]$.

Such divergent adversaries on the probabilistic region graph $[\mathcal{S}]$ correspond to an infinite number of adversaries on the concurrent probabilistic system $\mathcal{S}$, some of which will be divergent in the sense of Definition 4.4.6. Conversely, for any divergent adversary of $\mathcal{S}$, there exists a corresponding divergent adversary on $[\mathcal{S}]$. These facts follow from Lemma 5.4.7 and Lemma 5.4.8.

5.5 Model checking using the probabilistic region graph

A method for model checking probabilistic timed automata against PTCTL formulae, which makes use of the probabilistic region graph, is now presented. Our approach proceeds in three steps:

1. construction of the probabilistic region graph as a finite state representation of the probabilistic timed automaton in question;

2. translating a formula of PTCTL into a formula of (a marginally extended version of) PBTL; and,

3. resolving this new formula on the probabilistic region graph using established PBTL model checking techniques [BdA95, BK98].

Subformula of ϕ	Subformula of Φ
`true`	`true`
a	a
ζ	a_ζ
$\phi_1 \wedge \phi_2$	$\Phi_1 \wedge \Phi_2$
$\neg\phi$	$\neg\Phi$
$z.\phi$	$z.\Phi$
$[\phi_1 \exists\mathcal{U}\, \phi_2]_{\sqsupseteq\lambda}$	$[\Phi_1 \exists\mathcal{U}\, \Phi_2]_{\sqsupseteq\lambda}$
$[\phi_1 \forall\mathcal{U}\, \phi_2]_{\sqsupseteq\lambda}$	$[\Phi_1 \forall\mathcal{U}\, \Phi_2]_{\sqsupseteq\lambda}$

Table 5.1: Rules for the derivation of a PBTL^z formula from a PTCTL formula.

As we have already seen how to construct the probabilistic region graph $[\mathcal{S}_T^\phi]$ of a probabilistic timed automaton T with respect to a PTCTL formula ϕ in Section 5.4, we first consider point 2, and present an extended variant of PBTL which can be used for model checking $[\mathcal{S}_T^\phi]$. The variant, termed PBTL^z, simply adds a reset quantifier expression to PBTL, which, naturally, is used to represent the reset quantifier of PTCTL. Note that PBTL^z uses the extended set [AP] of atomic propositions.

Definition 5.5.1 (Syntax of PBTL^z**)** *The syntax of* PBTL^z *is defined as follows:*

$$\Phi ::= \texttt{true} \;\mid\; a \;\mid\; \Phi \wedge \Phi \;\mid\; \neg\Phi \;\mid\; z.\Phi \;\mid\; [\Phi \exists\mathcal{U}\, \Phi]_{\sqsupseteq\lambda} \;\mid\; [\Phi \forall\mathcal{U}\, \Phi]_{\sqsupseteq\lambda}$$

where $a \in [\mathrm{AP}]$ is an atomic proposition, $z \in \mathcal{Z}$, $\lambda \in [0,1]$, and $\sqsupseteq \in \{\geq, >\}$.

The satisfaction relation of PBTL^z is obtained from that of PBTL, presented in Definition 3.3.2, with the addition of an extra rule concerning the reset quantifier. As regions are the states of the concurrent probabilistic system $[\mathcal{S}_T^\phi]$, we write $s[z := 0]$ for the region $(v, \alpha[z := 0])$ if $s = (v, \alpha)$.

Definition 5.5.2 (Satisfaction Relation for PBTL^z**)** *Given the probabilistic region graph $[\mathcal{S}_T^\phi]$ for a probabilistic timed automaton T and a PTCTL formula ϕ, and a set $\mathcal{B}$ of adversaries of $[\mathcal{S}_T^\phi]$, then for any state $s \in [S]$ of $[\mathcal{S}_T^\phi]$ and a PBTL^z formula Φ, the satisfaction relation is defined according to the rules in Definition 3.3.2 and the following:*

$$s \models_{\mathcal{B}} z.\Phi \;\Leftrightarrow\; s[z := 0] \models_{\mathcal{B}} \Phi$$

Furthermore, a PBTL^z formula, Φ, can be *derived from* a PTCTL formula, ϕ, by applying the rules in Table 5.1 inductively. Now we can use the model checking algorithm of [BdA95, BK98] in order to verify whether the PBTL^z formula Φ holds in an initial state of the probabilistic region graph $[S_T^\phi]$.

Proposition 5.5.3 (Correctness of the model checking procedure) *Let T be a probabilistic timed automaton and ϕ be a PTCTL formula. Then a state $(v, \mathbf{a}) \in S$ of the concurrent probabilistic system $\mathcal{S}_T^\phi$ satisfies ϕ over divergent adversaries if and only if the region $(v, [\mathbf{a}]) \in [S_T^\phi]$ of the probabilistic region graph $[\mathcal{S}_T^\phi]$ satisfies the PBTL^z formula Φ interpreted over divergent adversaries, where Φ is derived from ϕ.*

Proof. Before considering any temporal or probabilistic operators, we must be convinced that subformulae comprising only atomic propositions, zones, boolean connectives and reset quantifiers are resolved satisfactorily. We proceed to show this by induction on the structure of ϕ.

If $\phi = \texttt{true}$, then ϕ will be true for all states of $\mathcal{S}_T^\phi$ and all formula clock valuations. Here ϕ derives $\Phi = \texttt{true}$, which is true for all regions in $[S_T^\phi]$.

If $\phi = a$, where $a \in \mathrm{AP}$, then it is true for state $(v, \mathbf{a})$ of $\mathcal{S}_T^\phi$ if and only if $a \in L(v, \mathbf{a})$. By Definition 4.3.2, we have $a \in L(v, \mathbf{a})$ if and only if $a \in \mathcal{L}(v)$. By Definition 5.4.2, $a \in [L](v, [\mathbf{a}])$ if $a \in \mathcal{L}(v)$, so $\Phi = a$ is true for the region $(v, [\mathbf{a}])$.

If $\phi = \zeta$, then the state $(v, \mathbf{a})$ of $\mathcal{S}_T^\phi$ satisfies ζ if $\mathbf{a} \in \zeta$. Then, from Definition 5.4.2, $a_\zeta \in [L](v, [\mathbf{a}])$. Because $\Phi = a_\zeta$, and Φ is derived from ϕ, both ϕ and Φ resolve to true in $(v, \mathbf{a})$ and $(v, [\mathbf{a}])$ respectively.

The cases of the boolean connectives, $\neg$ and $\wedge$, are self-evident.

If $\phi = z.\phi'$, then, for a given state $(v, \mathbf{a})$ that satisfies ϕ', we know that the region $(v, [\mathbf{a}])$ will also satisfy $z.\Phi'$, by observing the following argument. For any sets of adversaries $\mathcal{A} \subseteq Adv, \mathcal{B} \subseteq [Adv]$, then by Definition 5.2.3:

$$\begin{array}{rcll} (v, \mathbf{a}) \models_{\mathcal{A}} z.\phi' & \Leftrightarrow & (v, \mathbf{a}) \models_{\mathcal{A}} \phi & \\ & \Leftrightarrow & (v, [\mathbf{a}[z := 0]]) \models_{\mathcal{B}} \Phi & \text{(by induction)} \\ & \Leftrightarrow & (v, [\mathbf{a}]) \models_{\mathcal{B}} z.\Phi' & \text{(by Definition 5.5.1).} \end{array}$$

Now we show that $(v, \mathbf{a}) \models_{Div} [\phi_1 \exists\mathcal{U}\, \phi_2]_{\sqsupseteq\lambda}$ if and only if $(v, [\mathbf{a}]) \models_{[Div]} [\Phi_1 \exists\mathcal{U}\, \Phi_2]_{\sqsupseteq\lambda}$. Our presentation is split into three sections.

1. Noting that, for a divergent path $\omega \in Path_{ful}^{\mathcal{S}_T^\phi}$ of $\mathcal{S}_T^\phi$, a corresponding divergent path $\pi \in Path_{ful}^{[\mathcal{S}_T^\phi]}$ of $[\mathcal{S}_T^\phi]$, can be constructed, and, given a divergent path $\pi \in Path_{ful}^{[\mathcal{S}_T^\phi]}$ of $[\mathcal{S}_T^\phi]$, we can construct a corresponding divergent path $\omega \in Path_{ful}^{\mathcal{S}_T^\phi}$ of $\mathcal{S}_T^\phi$, where the notion of correspondence is path inscription.

2. Showing that the prefixes of finite length of the two paths ω and π are associated with the same probability value.

3. Showing $\omega \models_{Div} \phi_1\mathcal{U}\phi_2$ if and only if $\pi \models_{[Div]} \Phi_1\mathcal{U}\Phi_2$.

We progress through each of these three requirements in turn.

1. Follows immediately from Lemma 5.4.7 and Lemma 5.4.8.

2. We must show that the probability value associated with each of the prefixes of finite length have the same probability for ω and π. We proceed by induction on the length of paths. Let $A \in \mathit{Div}$ and $B \in [\mathit{Div}]$ be the adversaries such that $\omega \in \mathit{Path}^A_{\mathit{ful}}$ and $\pi \in \mathit{Path}^B_{\mathit{ful}}$ (although note that the assumption of the divergence of adversaries is not relevant for this step).

For the base case, observe that $\mathit{Prob}^A_{\mathit{fin}}(\omega(0)) = \mathit{Prob}^B_{\mathit{fin}}(\pi(0)) = 1$.

Next, for every transition of ω, we identify $l+1$ transitions of π which correspond to this transition, for $l \in \mathbb{N}$. That is, for every segment $\omega_i = (v_i, \mathbf{a}_i) \xrightarrow{\delta_i} \xrightarrow{\theta_{p_i}, \bar{p}_i} (v_{i+1}, \mathbf{a}_{i+1})$ of ω, we identify the segment

$$\pi_i = (v_i, [\mathbf{a}_i]_0) \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_1) \xrightarrow{\tau} \cdots \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_{l_i}) \xrightarrow{\theta_{p_i}, \bar{p}_i} (v_{i+1}, [\mathbf{a}_{i+1}]_0)$$

of π, where $\mathbf{a}_i \in [\mathbf{a}_i]_0$, $[\mathbf{a}_i]_{l_i} \subseteq \mathit{pre}_{v_i}(p_i)$ and $\mathbf{a}_{i+1} \in [\mathbf{a}_{i+1}]_0$ (that such a segment can be identified follows from Lemma 5.4.7). Take $i \in \mathbb{N}$. By induction, we have established that $\mathit{Prob}^A_{\mathit{fin}}(\omega_0\omega_1...\omega_i) = \mathit{Prob}^B_{\mathit{fin}}(\pi_0\pi_1...\pi_i)$; our aim is now to show that $\mathit{Prob}^A_{\mathit{fin}}(\omega_0\omega_1...\omega_{i+1}) = \mathit{Prob}^B_{\mathit{fin}}(\pi_0\pi_1...\pi_{i+1})$. Recall that, by Definition 3.2.2, we have

$$\mathit{Prob}^A_{\mathit{fin}}(\omega_0...\omega_{i+1}) = \mathit{Prob}^A_{\mathit{fin}}(\omega_0...\omega_i) \cdot \mathbf{P}^A(\omega_0...\omega_i, \omega_0...\omega_{i+1}) \ ,$$

and similarly

$$\begin{aligned}
&\mathit{Prob}^B_{\mathit{fin}}(\pi_0...\pi_{i+1}) \\
=\ &\mathit{Prob}^B_{\mathit{fin}}(\pi_0...\pi_i)\cdot \\
&\mathbf{P}^B(\pi_0...\pi_i, \pi_0...\pi_i \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_1))\cdot \\
&\mathbf{P}^B(\pi_0...\pi_i \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_1), \pi_0...\pi_i \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_1) \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_2))\cdot \\
&... \cdot \\
&\mathbf{P}^B(\pi_0...\pi_i \xrightarrow{\tau} ... \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_{l_i}), \pi_0...\pi_i \xrightarrow{\tau} ... \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_{l_i}) \xrightarrow{\theta_{p_i}, \bar{p}_i} (v_{i+1}, [\mathbf{a}_{i+1}]_0)) \ .
\end{aligned}$$

This unwieldy equation can be reduced by observing the following fact: from Definition 5.4.2, we have

$$\mathbf{P}^B(\pi_0...\pi_i, \pi_0...\pi_i \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_1)) = \mathcal{D}(v_i, [\mathbf{a}_i]_1) = 1 \ ,$$

and

$$\mathbf{P}^B(\pi_0...\pi_i \xrightarrow{\tau} ... \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_j), \pi_0...\pi_i \xrightarrow{\tau} ... \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_{j+1})) = \mathcal{D}(v_i, [\mathbf{a}_i]_{j+1}) = 1$$

for all $1 \leq j < l_i$. Hence

$$\begin{aligned}
&\mathit{Prob}^B_{\mathit{fin}}(\pi_0...\pi_{i+1}) \\
=\ &\mathit{Prob}^B_{\mathit{fin}}(\pi_0...\pi_i)\cdot \\
&\mathbf{P}^B(\pi_0...\pi_i \xrightarrow{\tau} ... \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_{l_i}), \pi_0...\pi_i \xrightarrow{\tau} ... \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_{l_i}) \xrightarrow{\theta_{p_i}, \bar{p}_i} (v_{i+1}, [\mathbf{a}_{i+1}]_0)) \ .
\end{aligned}$$

Therefore, as $Prob^A_{fin}(\omega_0\omega_1...\omega_i) = Prob^B_{fin}(\pi_0\pi_1...\pi_i)$ by induction, in order to show that $Prob^A_{fin}(\omega_0\omega_1...\omega_{i+1}) = Prob^B_{fin}(\pi_0\pi_1...\pi_{i+1})$, we prove that

$$\begin{aligned}&\mathbf{P}^A(\omega_0...\omega_i, \omega_0...\omega_{i+1})\\ =\;& \mathbf{P}^B(\pi_0...\pi_i \xrightarrow{\tau} ... \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_{l_i}), \pi_0...\pi_i \xrightarrow{\tau} ... \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_{l_i}) \xrightarrow{\theta_{p_i}, \bar{p}_i} (v_{i+1}, [\mathbf{a}_{i+1}]_0))\ .\end{aligned}$$

First observe that, from Definition 4.3.2, we have

$$\mathbf{P}^A(\omega_0...\omega_i, \omega_0...\omega_{i+1}) = \tilde{p}_i(v_{i+1}, \mathbf{a}_{i+1})\ .$$

Secondly, from Definition 5.4.2, we have

$$\begin{aligned}&\mathbf{P}^B(\pi_0...\pi_i \xrightarrow{\tau} ... \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_{l_i}), \pi_0...\pi_i \xrightarrow{\tau} ... \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_{l_i}) \xrightarrow{\theta_{p_i}, \bar{p}_i} (v_{i+1}, [\mathbf{a}_{i+1}]_0))\\ =\;& \bar{p}_i(v_{i+1}, [\mathbf{a}_{i+1}]_0)\ .\end{aligned}$$

Then, by Lemma 5.4.9, we have $\tilde{p}_i(v_{i+1}, \mathbf{a}_{i+1}) = \bar{p}_i(v_{i+1}, [\mathbf{a}_{i+1}]_0)$, and therefore we have proved our required result.

3. Next we prove $\omega \models_{Div} \phi_1 \mathcal{U} \phi_2$ if and only if $\pi \models_{[Div]} \Phi_1 \mathcal{U} \Phi_2$. If $\omega(i) = (v_i, \mathbf{a}_i)$ for all $i \in \mathbb{N}$, then

$$\begin{aligned}&\omega \models_{Div} \phi_1 \mathcal{U} \phi_2\\
\Leftrightarrow\;& \exists \text{ position } (i,\delta) \text{ of } \omega \text{ such that } \omega(i)+\delta \models_{Div} \phi_2\\
&\text{and } \forall \text{ positions } (j,\delta') \text{ of } \omega \text{ such that } (j,\delta') \ll (i,\delta),\\
&\omega(j)+\delta' \models_{Div} \phi_1 \vee \phi_2 \qquad\qquad \text{(by Definition 5.2.3)}\\
\Leftrightarrow\;& \exists \text{ position } (i,\delta) \text{ of } \omega \text{ such that } (v_i, [\mathbf{a}_i+\delta]) \models_{[Div]} \Phi_2\\
&\text{and } \forall \text{ positions } (j,\delta') \text{ of } \omega \text{ such that } (j,\delta') \ll (i,\delta),\\
&(v_j, [\mathbf{a}_j+\delta']) \models_{[Div]} \Phi_1 \vee \Phi_2 \qquad\qquad \text{(by induction)}\\
\Leftrightarrow\;& \exists i' \in \mathbb{N} \text{ such that } \pi(i') \models_{[Div]} \Phi_2 \text{ and } \pi(j') \models_{[Div]} \Phi_1 \vee \Phi_2,\ \forall j' \leq i'\\
&\qquad\qquad \text{(by construction of } \pi)\\
\Leftrightarrow\;& \pi \models_{[Div]} \Phi_1 \mathcal{U} \Phi_2 \qquad\qquad \text{(by Definition 3.3.1).}\end{aligned}$$

We must again appeal to the notion of splitting ω and π into segments to see how the result of **2** can be used to show that the probability of the finite prefixes which determine whether ω and π satisfy $\phi_1 \mathcal{U} \phi_2$ and $\Phi_1 \mathcal{U} \Phi_2$, respectively, are equal. Say $\omega \models_{Div} \phi_1 \mathcal{U} \phi_2$ and $\pi \models_{[Div]} \Phi_1 \mathcal{U} \Phi_2$, and let (i,δ) be the *first* position of ω such that $\omega(i)+\delta \models_{Div} \phi_2$; hence, all positions (j,δ') of ω such that $(j,\delta') \ll (i,\delta)$ are such that $\omega(j)+\delta' \models_{Div} \phi_1 \vee \phi_2$. Then, by induction, we have $\pi_j(k) \models_{[Div]} \Phi_1 \vee \Phi_2$ for all $j \leq i$ and $k \leq l_i + 1$ (recall the definition of each segment π_j in point **2**). It follows that, since $\omega(i)+\delta \models_{Div} \phi_2$, there exists $m \leq l_i$ such that $\pi_i(m) \models_{[Div]} \Phi_2$, again by induction. Recall that we proved in **2** that $Prob^A_{fin}(\omega_0...\omega_i) = Prob^B_{fin}(\pi_0...\pi_i)$. Now consider the fact that

$$\begin{aligned}&\mathbf{P}^B(\pi_0...\pi_i, \pi_0...\pi_i \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_1)).\\
&\mathbf{P}^B(\pi_0...\pi_i \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_1), \pi_0...\pi_i \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_1) \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_2)).\\
&....\\
&\mathbf{P}^B(\pi_0...\pi_i \xrightarrow{\tau} ... \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_{l_i}), \pi_0...\pi_i \xrightarrow{\tau} ... \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_m)) \quad = \quad 1\ .\end{aligned}$$

Therefore

$$\begin{array}{l} \mathit{Prob}^B_{\mathit{fin}}(\pi_0 ... \pi_{i+1}). \\ \mathbf{P}^B(\pi_0 ... \pi_i, \pi_0 ... \pi_i \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_1)). \\ .... \\ \mathbf{P}^B(\pi_0 ... \pi_i \xrightarrow{\tau} ... \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_{l_i}), \pi_0 ... \pi_i \xrightarrow{\tau} ... \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_m)) \quad = \mathit{Prob}^B_{\mathit{fin}}(\pi_0 ... \pi_{i+1}) \ . \end{array}$$

Hence the probability of the finite prefixes $\omega_0 ... \omega_i$ and $\pi_0 ... \pi_i \xrightarrow{\tau} ... \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_m)$ are equal, and hence

$$\mathit{Prob}^A(\omega_0 ... \omega_i \uparrow) = \mathit{Prob}^B((\pi_0 ... \pi_i \xrightarrow{\tau} ... \xrightarrow{\tau} (v_i, [\mathbf{a}_i]_m)) \uparrow) \ .$$

Then, from the fact that for any divergent adversary $A \in \mathit{Div}$, we can find a divergent adversary $B \in [\mathit{Div}]$ such that all of their paths have required inscription properties, and from the result concerning the probability of finite paths of A and B satisfying $\phi_1 \mathcal{U} \phi_2$ and $\Phi_1 \mathcal{U} \Phi_2$, respectively, we conclude the following:

$$\begin{array}{rl} & \mathit{Prob}^A(\{\omega \in \mathit{Path}^A_{\mathit{ful}}(v, \mathbf{a}) \ \mid \ \omega \models_{\mathit{Div}} \phi_1 \ \mathcal{U} \ \phi_2\}) \\ = & \mathit{Prob}^B(\{\pi \in \mathit{Path}^B_{\mathit{ful}}(v, [\mathbf{a}]) \ \mid \ \omega \models_{[\mathit{Div}]} \Phi_1 \ \mathcal{U} \ \Phi_2\}) \ . \end{array}$$

□

Chapter 6

Reachability for probabilistic timed automata

6.1 Introduction

Although the verification technique of Chapter 5 can establish the correctness of a probabilistic timed automaton model against a broad class of properties, it suffers from potential inefficiency as a result of the high granularity involved in the region construction. The size of the probabilistic region graph is the same as that in the non-probabilistic case: that is, the number of its states is exponential in the number of clocks and the magnitude of the maximal constants with which they are compared to in zones, either in the system description or in the temporal logic formula [AD94]. However, consideration of a narrower class of properties than those of PTCTL allows us to adopt a different approach when constructing a finite-state representation of a probabilistic timed automaton.

Here, we extend the real-time reachability properties of [DOTY96, LPY97a] with probability to obtain *probabilistic real-time reachability properties*. We are therefore interested in properties such as, "with probability 0.9999 or greater, it is possible to correctly deliver a data packet" from the example in Section 4.2.3. It will also be shown how to represent invariance properties, such as "with probability 0.875 or greater, the system never aborts", and time bounded reachability properties, such as "with probability 0.975 or greater, it is possible to correctly deliver a data packet within 5 time units", as reachability properties.

As for the case of non-probabilistic, real-time reachability, our finite-state model derived from the probabilistic timed automaton T is obtained, not by the region construction, but by *forward* search through the infinite state space of T. Once this has been done, *probabilistic reachability* analysis is then performed through computation of probabilities involving linear programming using well-established techniques [BdA95, BK98, dA97a, dA99b] (in the same way as linear programming is used in probabilistic model checking for quantitative temporal logic properties). We introduce the reachability algorithm in Section 6.2 and Section 6.3, and, in Section 6.4, give an example of its application to the communication protocol of Section 4.2.3. Section 6.5 considers an approach to progress within the forward reachability framework. In Section 6.6, it will be shown that the maximal probability of reaching the target set of states computed for the finite-state representation is an *upper bound* on the actual maximal probability of the probabilistic timed automaton T reaching the target state set. We conclude that our technique is less appropriate for the verification of reachability properties, for which we wish to establish whether the maximal probability of reaching a target set of states exceeds some bound, than invariant properties, for which we wish to establish whether the maximal probability of reaching certain "unsafe" states is less than some bound. Note that material from this chapter also appears in [KNSS00a].

For simplicity, as in Chapter 5, we assume that the post conditions of a probabilistic timed automaton can only set the value of a clock to 0, rather than to an arbitrary integer value, and that the initial state is of the form $(\bar{v}, \mathbf{0})$.

6.2 Specification of reachability properties

We now explain how probabilistic real-time reachability properties of probabilistic timed automata can be specified. We formally define such properties, and introduce the way in which their observational power may be increased by syntactic adjustments to the probabilistic timed automaton in question. Furthermore, two important subclasses of reachability properties are identified, and the correspondence between all probabilistic real-time reachability properties and PTCTL properties is identified.

6.2.1 Reachability of locations

We first focus on probabilistic real-time reachability properties for which the target set is described in terms of a set of locations. In the following, we fix the probabilistic timed automaton $T = (\mathcal{X}, V, \mathcal{L}, \mathit{init}, \mathit{inv}, \mathit{prob}, \langle \mathit{pre}_v \rangle_{v \in V})$. Henceforth, we assume that all of the probabilistic timed automata in this chapter have *sufficient deadlock freedom* (see Definition 4.4.9 in Section 4.4.2), unless explicitly stated otherwise. We explain the implications of this assumption in Section 6.3.2.

Definition 6.2.1 (Reachability problem) *For the probabilistic timed automaton T, let $\mathit{Target} \in V$ be a set of locations of T called the* target set*, let $\sqsupseteq \in \{\geq, >\}$, and let $\lambda \in [0, 1]$ be the* target probability*. Then a* probabilistic real-time reachability problem *for T is defined as the triple $(\mathit{Target}, \sqsupseteq, \lambda)$.*

*Let $\mathcal{A} \in Adv$ be a set of adversaries of the concurrent probabilistic system $\mathcal{S}_T$ of T. Then the answer to the problem $(Target, \sqsupseteq, \lambda)$ with respect to $\mathcal{A}$ is "*Yes*" if and only if there exists an adversary $A \in \mathcal{A}$ such that*

$$Prob^A(\{\omega \in Path^A_{ful}(\bar{v}, \mathbf{0}) \mid \exists i \in \mathbb{N}.\mathsf{discrete}(\omega(i)) \in Target\}) \sqsupseteq \lambda \ ,$$

*and "*No*" otherwise.*

Intuitively, the answer to the problem is "Yes" if and only if T can reach a state in the target set *Target* with probability $\sqsupseteq \lambda$, and "No" otherwise. That is, the answer is "Yes" if and only if an adversary in the set $\mathcal{A}$ for which the probability measure over paths which reach a location in *Target* is $\sqsupseteq \lambda$ can be found.

Observe that probabilistic real-time reachability is considered over *infinite* paths, whereas non-probabilistic reachability for timed automata is traditionally defined over finite paths [Tri98]. To see why this is necessary, consider again the probabilistic timed automaton T_{time} in Figure 4.7. It is clear that, for any discrete-progress adversary of $\mathcal{S}_{T_{\text{time}}}$, the probability of reaching v_2 is 1; however, we need the information about *infinite paths* such as ω_{time} (more precisely, the information that they are exhibited with probability 0) to conclude this.

The class of reachability problems for which $\sqsupseteq = \geq$ and $\lambda = 0$ is meaningless from a specification perspective (such properties are trivially true unless there does not exist an adversary of the probabilistic timed automaton at all, in which case a modelling error has occurred), and therefore we do not consider it here. The property of Section 4.2.3, "with probability 0.999 or greater, it is possible to correctly deliver a data packet", can be represented as the reachability property $(\{\mathsf{ri}\}, \geq, 0.999)$. That is, the target set is the single location ri, and the target probability is 0.999.

6.2.2 Reachability of symbolic states

As in the non-probabilistic context [Tri98], we can expand the scope of our properties from the reachability of locations to the reachability of *symbolic states*. First we explain what a symbolic state is.

Definition 6.2.2 (Symbolic state) *For the probabilistic timed automaton T, a pair of the form (v, ζ), where $v \in V$ and $\zeta \in \mathsf{Zone}(\mathcal{X})$ is a zone such that $\zeta \subseteq inv(v)$, is called a* symbolic state.

We say that a state $(v, \mathbf{a})$ is *contained within* a symbolic state (v', ζ), written $(v, \mathbf{a}) \in (v', \zeta)$, if $v = v'$ and $\mathbf{a} \in \zeta$. Because any valuation is a zone, any state is also a symbolic state. We overload the function $\mathsf{discrete}$ of Section 4.3 from states to symbolic states: for a symbolic state (v, ζ), we define $\mathsf{discrete}(v, \zeta) = v$.

We now define formally the probabilistic real-time reachability problem for symbolic states.

Definition 6.2.3 (Reachability problem for symbolic states) *For the probabilistic timed automaton T, let $Target_{\mathrm{symb}} = \{(v_1, \zeta_1), ..., (v_m, \zeta_m)\}$ be a set of symbolic states of T called the* symbolic target set*, let $\sqsupseteq \in \{\geq, >\}$, and let $\lambda \in [0,1]$ be the target probability. Then the* probabilistic real-time reachability problem for symbolic states *is the triple $(Target_{\mathrm{symb}}, \sqsupseteq, \lambda)$.*

We do not include a formal definition of how to obtain the answer to reachability problems for symbolic states, because each such problem can be reduced to a standard probabilistic real-time reachability problem without loss of generality. The method is based on the addition of extra transitions to T which are enabled when the target symbolic state is reached, and lead to an added location v', which we then regard as the target location of a standard reachability problem.

Formally, we reduce the probabilistic real-time reachability problem for symbolic states of T, denoted by $(Target_{\mathrm{symb}}, \sqsupseteq, \lambda)$, to the standard real-time reachability problem on a *new* probabilistic timed automaton T' in the following manner. Recall that, for a location v, the distribution $\mathcal{D}(v, X)$ denotes the Dirac distribution over a single edge with the target location of v, and the reset set of X. Let

$$T' = (\mathcal{X}, V \cup \{v'\}, \mathcal{L}', init, inv', prob', \langle pre'_v \rangle_{v \in V \cup \{v'\}})$$

where:

- $\mathcal{L}'(v') = \emptyset$, $inv'(v') = \texttt{true}$, $prob'(v') = \{\mathcal{D}(v', \emptyset)\}$, and $pre'_{v'}(\mathcal{D}(v', \emptyset)) = \texttt{true}$,
- for all locations $v \in V$, let $\mathcal{L}'(v) = \mathcal{L}(v)$, and $inv'(v) = inv(v)$. Furthermore, let

  $$prob'(v) = prob(v) \cup \{\mathcal{D}(v', \emptyset)_{(v,\zeta)} \mid (v, \zeta) \in Target_{\mathrm{symb}}\} ,$$

  where $pre'_v(\mathcal{D}(v', \emptyset)_{(v,\zeta)}) = \zeta$, and $pre'_v(p) = pre_v(p)$ for all other $p \in prob(v)$.

The reachability problem for symbolic states $(Target_{\mathrm{symb}}, \sqsupseteq, \lambda)$ on T is then reduced to the associated standard reachability problem $(\{v'\}, \sqsupseteq, \lambda)$ on T'. The answer to the problem $(Target_{\mathrm{symb}}, \sqsupseteq, \lambda)$ on T is "YES" if the answer to the problem $(\{v'\}, \sqsupseteq, \lambda)$ on T' is "YES", and is "NO" otherwise.

6.2.3 Classification of reachability properties

We now identify two subclasses of reachability properties which are particularly relevant for the verification of probabilistic real-time systems.

Class 1: time bounded reachability

In certain cases, we may be interested in the reachability of certain locations either before or after a time deadline has expired. For example, recall the communication protocol of Section 4.2.3, and consider the property, "with probability 0.975 or greater, it is possible to correctly deliver a data packet within 5 time units". Problems of this type can be solved within the framework of reachability in the following manner. First, the probabilistic timed automaton of interest, T, is augmented with a single clock z, which is intended to count the total elapsed time of system execution. The clock z does not feature in any of the invariant or enabling conditions, or in any clock resets of the new probabilistic timed automaton, which will be denoted by T_{+z} (although observe that, as usual, z is 0 initially). Say that we are interested in the reachability of the set *Target* of locations in time $\sim c$, where $\sim \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$; that is, the probabilistic time bounded reachability problem is described by the tuple $((\mathit{Target}, \sim, c), \sqsupseteq, \lambda)$. Then this problem can be phrased as the standard reachability problem $(\mathit{Target}^{\sim c}_{\text{symb}}, \sqsupseteq, \lambda)$ on symbolic states, where

$$\mathit{Target}^{\sim c}_{\text{symb}} = \{(v, z \sim c) \mid v \in \mathit{Target}\} ,$$

and where $z \sim c$ is the syntactic representation of the required zone. The probabilistic timed automaton T_{+z} then undergoes a further transformation to T'_{+z} in the manner described in the previous section, on which the standard probabilistic real-time reachability analysis is then performed. Then the answer to the time bounded reachability problem $((\mathit{Target}, \sim, c), \sqsupseteq, \lambda)$ on T is "Yes" if the answer the reachability problem of symbolic states $\{(v, z \sim c) \mid v \in \mathit{Target}\}$ on T_{+z} is "Yes", and is "No" otherwise.

Class 2: invariance verification

Another type of property of interest requires that the probabilistic timed automaton T does not leave a certain set of locations $\mathit{Invariant} \subseteq V$ with a given probability or greater. For example, in Section 4.2.3, the property, "with probability 0.875 or greater, the system never aborts", falls into this category, where $\mathit{Invariant} = V \setminus \{\mathsf{aa}\}$. Such *invariance verification* properties can be reduced to standard probabilistic real-time reachability problems in the following way. Let the probabilistic invariance verification problem $\mathcal{I}$ be the tuple $(\mathit{Invariant}, \sqsupseteq, \lambda)$, where $\sqsupseteq$ and λ have their usual meanings. Then $\mathcal{I}$ reduces to the probabilistic real-time reachability problem $(V \setminus \mathit{Invariant}, \bar{\sqsupseteq}, 1 - \lambda)$, where $\bar{\sqsupseteq} => $ if $\sqsupseteq = \geq$ and $\bar{\sqsupseteq} = \geq$ if $\sqsupseteq = >$. That is, we solve the problem of reachability of locations that are *not* in the required set *Invariant* with probability $1 - \lambda$ or greater. If the answer to the problem $(V \setminus \mathit{Invariant}, \bar{\sqsupseteq}, 1 - \lambda)$ is "Yes", then the answer to $\mathcal{I}$ is "No"; otherwise the answer to $\mathcal{I}$ is "Yes".

6.2.4 Expressiveness of reachability properties

Consider the fact that probabilistic real-time reachability properties require that it is *possible* to reach a certain set of states with probability λ or *greater*. It then follows that we are able to verify the dual of such properties; more precisely, those specifying the *inevitability* of reaching a certain set of states with a given probability λ or *less*. Clearly, such a property is true if it is not possible to reach the set of states with probability greater than λ.

We now consider the way in which certain PTCTL properties are equivalent to reachability properties, given certain assumptions about the probabilistic timed automaton on which the properties are interpreted. In the non-probabilistic context, property specification languages such as TCTL do not capture reachable states [Tri98], for the simple reason that the former are evaluated over infinite, divergent paths, whereas the latter are defined by finite paths. This characteristic also transfers to the probabilistic context, albeit in a slightly different manner; that is, formulae of the logic PTCTL are interpreted over divergent adversaries, whereas, as we will see in Section 6.5, reachability properties are interpreted over a *superset* of divergent adversaries. However, if *all* adversaries of the concurrent probabilistic system $\mathcal{S}_T$ of T are divergent, then we can identify correspondences between PTCTL formulae and reachability properties. Without loss of generality, for a given set $U \subseteq V$ of locations, we extend the labelling condition $\mathcal{L}$ of the probabilistic timed automaton T to $\mathcal{L}_U : V \to 2^{\mathrm{AP} \cup \{a_U\}}$ where, for each $v \in V$, $\mathcal{L}_{Target}(v) = \mathcal{L}(v) \cup \{a_U\}$ if $v \in U$, and $\mathcal{L}_{Target}(v) = \mathcal{L}(v)$ otherwise.

Assume that all adversaries of the concurrent probabilistic system $\mathcal{S}_T$ of T are divergent in the manner of Definition 4.4.6. Then the reachability problem $(Target, \sqsupseteq, \lambda)$ is equivalent to the PTCTL formula $[\mathtt{true}\ \exists\mathcal{U}\ a_{Target}]_{\sqsupseteq\lambda}$ (which can be abbreviated to $[\exists\Diamond a_{Target}]_{\sqsupseteq\lambda}$). That is, $(Target, \sqsupseteq, \lambda)$ is true if and only if there exists a (divergent) adversary such that the probability of reaching a location in *Target* is $\sqsupseteq \lambda$. Similarly, the probabilistic time bounded reachability problem $((Target, \sqsupseteq, \lambda), \sim, c)$ can be restated as the PTCTL formula $z.[\mathtt{true}\ \exists\mathcal{U}\ (a_{Target} \wedge z \sim c)]_{\sqsupseteq\lambda}$ (which can be abbreviated to $z.[\exists\Diamond(a_{Target} \wedge z \sim c)]_{\sqsupseteq\lambda}$). Dually, the invariance verification problem $(Invariant, \sqsupseteq, \lambda)$ is equivalent to the PTCTL formula $\neg[\mathtt{true}\ \exists\mathcal{U}\ \neg a_{Invariant}]_{\bar{\sqsupseteq}1-\lambda}$ (which can be abbreviated to $[\forall\Box a_{Invariant}]_{\sqsupseteq\lambda}$). In other words, $(Invariant, \sqsupseteq, \lambda)$ is true if and only if there does not exist a (divergent) adversary which leaves *Invariant* with probability $\bar{\sqsupseteq}1-\lambda$.

6.3 Probabilistic real-time reachability algorithm

6.3.1 Preliminaries

Before the probabilistic real-time reachability algorithm is introduced formally, we present the following notation to represent and manipulate the state sets computed during the algorithm. The concepts *c-equivalence* and *c-closure* are defined in a similar manner to [Tri98] (note that they are related to the "extrapolation abstraction" of [DT98]).

Definition 6.3.1 (c-equivalence and c-closure cf. [Tri98]) *Given $c \in \mathbb{N}$, the two* *valuations* $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n_{\geq 0}$ *are described as* c-equivalent *if:*

- *for any clock* $x_i \in \mathcal{X}$*, either* $\mathbf{a}_i = \mathbf{b}_i$*, or* $\mathbf{a}_i > c$ *and* $\mathbf{b}_i > c$*; and,*
- *for any two clocks* $x_i, x_j \in \mathcal{X}$*,* $\mathbf{a}_i - \mathbf{a}_j = \mathbf{b}_i - \mathbf{b}_j$*, or* $\mathbf{a}_i - \mathbf{a}_j > c$ *and* $\mathbf{b}_i - \mathbf{b}_j > c$*.*

We define the c-closure *of* ζ*, denoted by* $\mathsf{close}(\zeta, c)$*, to be the greatest zone* $\zeta' \supseteq \zeta$ *satisfying, for all* $\mathbf{b} \in \zeta'$*, there exists* $\mathbf{a} \in \zeta$ *such that* $\mathbf{a}$ *and* $\mathbf{b}$ *are c-equivalent.*

Intuitively, the c-closure of a zone is obtained by removing all of its boundaries which correspond to constraints which refer to integers greater than c. Observe that, for a given c, there are only a finite number of c-closed zones. The function close is extended to symbolic states, by defining $\mathsf{close}((v, \zeta), c) = (v, \mathsf{close}(\zeta, c))$.

Our aim of forward exploration through the state space of a probabilistic timed automaton requires operations to return the successor states of all of the states in a particular set (where the set is represented as a symbolic state). More precisely, we introduce a *discrete successor* operation which, given an edge $e \in E$ and a symbolic state (v, ζ), returns all of the states obtained by traversing e from a state in (v, ζ). Similarly, our *time successor* operation on the symbolic state (v, ζ) returns the set of states which can be obtained from a state in (v, ζ) by letting time elapse. These two operations can be composed to define a generalised successor operation, called the *post* operation. For a given symbolic state (v, ζ) and an edge $e \in E$, the post operation returns the set of states that can be obtained from (v, ζ) by traversing the edge e and then letting time elapse. Note that we also parameterise the post operation by an integer $c \in \mathbb{N}$, and only compute the c-closure of symbolic states obtained by the time successor operation; in Section 6.3, this fact will be used to ensure the termination of our forward reachability algorithm.

Definition 6.3.2 (Successor operations) *Let* (v, ζ) *be a symbolic state of the probabilistic timed automaton* T*.*

Time successor *The* time successor *of* (v, ζ) *is defined as*

$$\mathsf{time\_succ}(v, \zeta) \overset{\text{def}}{=} (v, \nearrow\!\zeta \cap inv(v)) \ .$$

Discrete successor *The* discrete successor *of* (v, ζ) *with respect to the edge* $e = (v, v', X, p)$ *is defined as*

$$\mathsf{disc\_succ}(e, (v, \zeta)) \overset{\text{def}}{=} (v', ((\zeta \cap pre_v(p))[X := 0]) \cap inv(v')) \ .$$

Post *For a constant* $c \in \mathbb{N}$ *and an edge* $e \in \mathsf{out}(v)$*, the* post operation *is defined according to:*

$$\mathsf{post}[e, c](v, \zeta) \overset{\text{def}}{=} \mathsf{close}(\mathsf{time\_succ}(\mathsf{disc\_succ}(e, (v, \zeta)), c)) \ .$$

Observe that all of the operations in Definition 6.3.2 preserve the convexity of zones (they are essentially the same operations as presented in [Tri98], in which it is shown formally that the operations preserve convexity), and therefore they always return symbolic states.

6.3.2 The algorithm ForwardReachability

An algorithm for generating a finite representation of the state space of a probabilistic timed automaton which is relevant to a given set of target locations is presented in Figure 6.1. As in the case of similar algorithms in the non-probabilistic context [DOTY96, LPY97a], the algorithm searches forward through a reachable portion of the state space of the system by iterating successively the transition relation a finite number of times. Note that the introduction of probabilistic information in the discrete transition relation of probabilistic timed automata means that we now regard the portion of the state space that is computed by this method to be reachable *with non-zero probability.* Given that this set has been generated for the probabilistic timed automaton T, we can then obtain an upper bound on the maximal probability of computations of T reaching the target set; this is achieved by solving a linear programming problem on the generated state space, in the manner of [dA97a, dA99b]. Therefore, our strategy consists of *two* distinct computation steps: firstly, generation of the state space which is reachable with non-zero probability from the initial state, and secondly, performing another computation on this state space to find the maximum probability relevant to our problem.

Generation of the state space

The finitary representation of the portion of the state space reachable with nonzero probability consists of a finite number of state sets, which, unlike in the probabilistic region graph representation, are not necessarily disjoint. The state sets are generated by the post operation presented in Definition 6.3.2, and therefore take the form of symbolic states; that is, they are obtained by alternating the discrete successor relation with the time successor relation. Each such symbolic state is associated with a finite set of distributions, with the elements in the support of each distribution being a symbolic state. Therefore, we regard this representation as being a concurrent probabilistic system, the states of which are the computed symbolic states, and the transitions of which are the distributions over symbolic states. We observe that the transitions of this concurrent probabilistic system correspond to *discrete* transitions of the probabilistic timed automaton only; unlike in the probabilistic region graph of Chapter 5, time transitions are abstracted completely.

One important difference between the algorithm in Figure 6.1 and analogous algorithms in the non-probabilistic, real-time literature, is the fact that an appealing *early termination* property of the latter algorithms is compromised in our context. This property refers to the fact that, if a symbolic state which reaches the target set is computed, then the algorithm can terminate immediately with a "Yes" answer to the reachability problem. Such a strategy is insufficient for probabilistic timed automata. For example, consider the case in which we have found a path of symbolic states reaching the target set *Target* which corresponds to the probability λ. Then it may be possible to find another path to *Target*, thus increasing the probability of reaching the target set. Given the expense of the probability computation step, we opt to perform it only *after* the forward exploration algorithm has terminated.

```
ForwardReachability($(\bar{v}, \mathbf{0})$, $Target$){
  $c := c_{\max}(T)$
  $SymbStates := \emptyset$
  $Reached := \emptyset$
  $Fringe := \{\mathsf{close}(\mathsf{time\_succ}(\bar{v}, \mathbf{0}), c)\}$
  repeat
    choose $(v, \zeta) \in Fringe$
    $Fringe := Fringe \setminus \{(v, \zeta)\}$
    if $v \in Target$ then $Reached := Reached \cup \{(v, \zeta)\}$
    else
      for each $e \in \mathsf{out}(v)$ do
        let $(v', \zeta') := \mathsf{post}[e, c](v, \zeta)$
        if $\zeta' \neq \emptyset$ and $(v', \zeta') \notin SymbStates$ then
          $Fringe := Fringe \cup \{(v', \zeta')\}$
        end if
      end for each
    end if
    $SymbStates := SymbStates \cup \{(v, \zeta)\}$
  until $Fringe = \emptyset$
  return $SymbStates, Reached$
}
```

Figure 6.1: The algorithm **ForwardReachability**

The algorithm **ForwardReachability** takes as input the initial state $(\bar{v}, \mathbf{0})$ of the probabilistic timed automaton T, and then computes the *initial symbolic state*, denoted by $\mathsf{close}(\mathsf{time\_succ}(\bar{v}, \mathbf{0}), c)$, by taking the time successor of the initial state, and then computing its c-closure, where c is equal to the maximal constant $c_{\max}(T)$ used in the description of T. The algorithm also takes a target set *Target* of locations as input. At any point during the execution of the algorithm, the set *SymbStates* denotes the set of symbolic states that have been generated by iterating the post operation from the initial symbolic state, *and* have had their successor symbolic states generated. Therefore, once a symbolic state is contained in the set *SymbStates*, we do not need to apply the post operation to it again. The set *Reached* is a subset of *SymbStates* which contains all of the symbolic states generated by the algorithm for which the location component is in the target set *Target*. Finally, the set *Fringe* contains the symbolic states for which the successors have not yet been generated. Therefore, initially *Fringe* contains the time successor symbolic state of the initial state $(\bar{v}, \mathbf{0})$, and the condition of the algorithm's **repeat until** loop is that the set *Fringe* is non-empty (corresponding to the case in which there are still more successor symbolic states to explore).

The commands within the body of the **repeat until** loop take the following form. For each iteration of this loop, a symbolic state is extracted from the set *Fringe*; if its location component is in the target set *Target*, then the symbolic state is added to the set *Reached*. Otherwise, for each edge with the location of the symbolic state as its source, a successor symbolic state is computed using the operation post; if such a successor has a non-empty zone and is not already in the set of already explored states *SymbStates*, then it is added to the set *Fringe*. The original symbolic state is then added to *SymbStates*.

Observe that, if a symbolic state extracted from *Fringe* has as its location component a location in *Target*, then its successors are *not* computed. This is so that, given that we have discovered a sequence of symbolic states from the initial state to the target, we do not continue to compute the behaviour of the system *after* the target set has been reached. Although this strategy can reduce the computation necessary to yield an answer to the reachability problem, it has unfortunate consequences for the study of progress with regard to reachability, as we will discuss in Section 6.5.

As a final point on the algorithm **ForwardReachability**, consider the line,

$$\textbf{let } (v', \zeta') := \mathsf{post}[e, c](v, \zeta) \; .$$

The assumption of sufficient deadlock freedom guarantees that, for any symbolic state (v, ζ) computed using post operations from the initial state, and an edge $e \in \mathsf{out}(v)$, then $(v', \zeta') := \mathsf{post}[e, c_{\max}(T)](v, \zeta)$ is such that $\zeta' \neq \emptyset$. We present an informal argument to support this claim: first assume that $\zeta \subseteq \mathsf{free}(v)$. Then, by the definition of $\mathsf{free}(v)$, this means that taking the forward diagonal projection of ζ, which corresponds to letting any amount of time pass, results in a zone which has a non-empty intersection with the pre-condition of a distribution $p \in \mathit{prob}(v)$, which in turn is intersected with the invariant condition. Say we then take the edge $(v, v', X, p) \in \mathsf{out}(v)$, and compute the next symbolic state as $(v', \zeta') = \mathsf{post}[e, c_{\max}(T)](v, \zeta)$. Then, by Definition 4.4.9, it must be the case that $\zeta' \subseteq \mathsf{free}(v')$. The case for the initial location $\bar{v}$ follows naturally. Therefore, the assumption of sufficient deadlock freedom guarantees that every computed symbolic state will have at least one successor symbolic state.

Computation of the maximal reachability probability

We now turn out attention to the way in which an upper bound on the maximum probability of the probabilistic timed automaton T reaching the target set of locations *Target* can be computed. Given the probabilistic timed automaton T and the target set $\mathit{Target} \subseteq V$, we obtain the sets *SymbStates*, *Reached* using the algorithm **ForwardReachability**. These sets are now used to define a concurrent probabilistic system $\mathcal{S}_T^{\mathit{Target}}$ called the *forward reachability graph*, which contains all of the information relevant to the probabilistic real-time reachability problem. The problem can then be solved on the state space of this system.

Definition 6.3.3 (Forward reachability graph) *The forward reachability graph for the probabilistic timed automaton T and the target set $\mathit{Target} \subseteq V$, is the concurrent probabilistic system $\mathcal{S}_T^{\mathit{Target}} = (\langle S \rangle, \langle \bar{s} \rangle, \langle L \rangle, \langle \Sigma \rangle, \langle \mathit{Steps} \rangle)$ defined in the following way.*

- *The state set $\langle S \rangle = \mathit{SymbStates}$ is the set of symbolic states computed by the algorithm* **ForwardReachability**.
- *The initial state $\langle \bar{s} \rangle \in \langle S \rangle$ is the symbolic state $\mathsf{close}(\mathsf{time\_succ}(\bar{v}, \mathbf{0}), c)$.*
- *The labelling function $\langle L \rangle$ is arbitrary.*
- *The event set is $\langle \Sigma \rangle = \{\theta_p \mid v \in V, p \in \mathit{prob}(v)\} \cup \{\bot\}$.*
- *The transition relation $\langle \mathit{Steps} \rangle : \langle S \rangle \to \langle \Sigma \rangle \times \mu(\langle S \rangle)$ is characterised in the following way. For each symbolic state $(v, \zeta) \in \langle S \rangle$, then $\langle \mathit{Steps} \rangle(v, \zeta)$ is the smallest set of event-distribution pairs satisfying:*
  - *if $(v, \zeta) \in \mathit{SymbStates} \setminus \mathit{Reached}$, then for each distribution $p \in \mathit{prob}(v)$, where $\zeta \cap \mathit{pre}_v(p) \neq \emptyset$, there exists $(\theta_p, \hat{p}) \in \langle \mathit{Steps} \rangle(v, \zeta)$ such that, for each $(v', \zeta') \in \langle S \rangle$:*

$$\hat{p}(v', \zeta') = \sum_{\substack{X \subseteq \mathcal{X} \,\&\, \\ (v', \zeta') = \mathsf{post}[(v, v', X, p), c](v, \zeta)}} p(v', X) \; .$$

  - *if $(v, \zeta) \in \mathit{Reached}$, let $\langle \mathit{Steps} \rangle(v, \zeta) = \{(\bot, \mathcal{D}(v, \zeta))\}$.*

It is clear that, for any symbolic state $(v, \zeta) \in \mathit{SymbStates} \setminus \mathit{Reached}$, and for any distribution $p \in \mathit{prob}(v)$ which is such that $\zeta \cap \mathit{pre}_v(p) \neq \emptyset$, then we have $\mathsf{post}[(v, v', X, p), c](v, \zeta) \in \mathit{SymbStates}$ for each edge $e \in \mathsf{edge\_support}(p)$.

As in the standard manner for concurrent probabilistic systems (see Section 3.2.9), the notions of paths and adversaries apply to the forward reachability graph. We typically denote an adversary of $\mathcal{S}_T^{\mathit{Target}}$ by B, the set of all such adversaries by $\langle \mathit{Adv} \rangle$, and a subset of these adversaries by $\mathcal{B} \subseteq \langle \mathit{Adv} \rangle$. Then, the notion of the probability measure $Prob^B$ of an adversary $B \in \langle Adv \rangle$ can be defined in the standard manner of Definition 3.2.9.

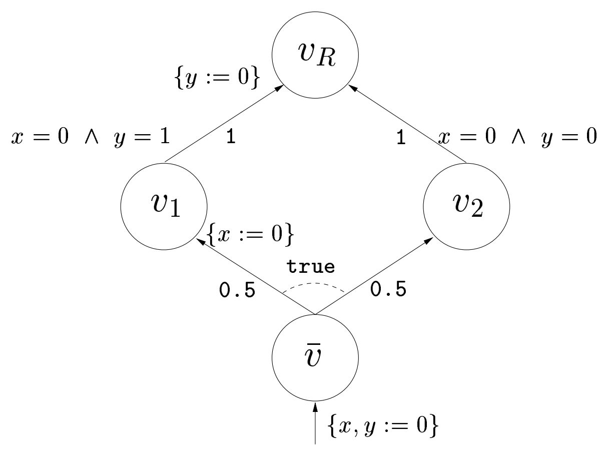
We now state that the forward reachability graph is finite.

Lemma 6.3.4 *For any probabilistic timed automaton T and target set $Target \subseteq V$, the concurrent probabilistic system $\mathcal{S}_T^{Target}$ is finite-state, and has finite nondeterministic branching.*

Recall that, for a finite $c \in \mathbb{N}$, there exist a finite number of c-closed zones, and observe that the new symbolic states are generated by the algorithm **ForwardReachability** using the operation post, which can only generate symbolic states with $c_{\max}(T)$-closed zones. Then, by Definition 4.2.1, because $c_{\max}(T)$ is finite, and the number of locations in V is also finite, it follows that the number of symbolic states generated by the algorithm is also finite. Noting that the termination of the **repeat until** loop of the algorithm depends on the emptiness of *Fringe*, which in turn depends on the new symbolic states that are generated by post operations, then the algorithm will terminate within a finite number of steps. The fact that $\mathcal{S}_T^{Target}$ has finite nondeterministic branching follows from the fact that, for each location $v \in V$, the set $prob(v)$ has finite cardinality.

Given that the forward reachability graph of the probabilistic timed automaton T with respect to the target set *Target* has been constructed, the probabilistic real-time reachability problem can be solved in the following way. First, observe that if $Reached = \emptyset$, then the forward search through the state space of T has found that *Target* is not reachable with positive probability, and therefore we conclude that the answer to the problem is "No". If $Reached \neq \emptyset$, we propose to perform a maximal reachability probability computation on the state space of the forward reachability graph, following established techniques for finite state Markov decision processes [BdA95, BK98, CY98, dA97a, dA99b]. Unfortunately, the probability obtained via this approach may be greater than the probability of reaching the target set via *any* adversary of the probabilistic timed automaton.

This is illustrated by the example of the probabilistic timed automaton T_{ex}, and its associated forward reachability graph shown in Figure 6.2. Say that the target set of interest consists of the single node v_R. Then it is clear that there are only two adversaries of T_{ex} (or, rather, of its concurrent probabilistic system $\mathcal{S}_{T_{ex}}$) such that there is a non-zero probability of reaching v_R. Intuitively, these correspond to two possible times at which the probability distribution associated with the initial state is chosen: either when $x = y = 0$, or when $x = y = 1$. In the former case, if the left-hand edge to v_1 is taken, then the outgoing edge of v_1 can never be taken, and so we must remain in this node; however, if the right-hand edge to v_2 is taken, then the outgoing edge to v_R can be selected immediately. The case for the selection of the transition of $\bar{v}$ when $x = y = 1$ is symmetric. Therefore, for each of the two adversaries



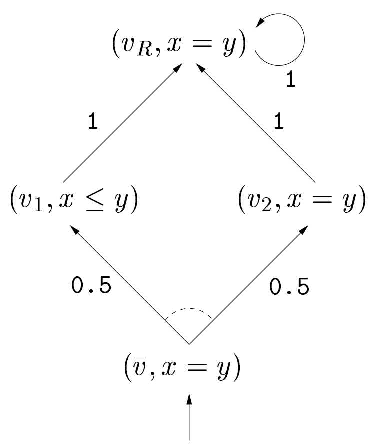


Figure 6.2: The probabilistic timed automaton T_{ex} and the forward reachability graph $\mathcal{S}^{v_R}_{T_{ex}}$.

that have been described informally, the probability of reaching the target node v_R is 0.5. Now consider the forward reachability graph $\mathcal{S}^{v_R}_{T_{ex}}$ shown in Figure 6.2 (the reader can verify that this is indeed the forward reachability graph corresponding to the reachability problem in question). If, in $(\bar{v}, x = y)$, the distribution to $(v_1, x \leq y)$ and $(v_2, x = y)$ is selected, and then, whichever symbolic state is chosen, a transition to $(v_R, x = y)$ is made, then the probability of reaching a symbolic state for which the discrete part is v_R is 1. If the probabilistic real-time reachability problem in question is $(\{v_R\}, \geq, 0.7)$, then the maximal reachability probability that is computed on the forward reachability graph will be greater than or equal to 0.7, whereas the answer to the probabilistic real-time reachability problem, as presented in Section 6.2, will be "No". Therefore, we advocate the strategy of returning an answer of "Maybe", rather than "Yes" if the maximal reachability probability computed on the forward reachability graph is $\sqsupseteq \lambda$, and "No" otherwise.

The maximal reachability probability problem on a Markov decision process is solved by reduction to an appropriate linear programming problem [BdA95, BK98, CY98, dA97a, dA99b]. We now apply this solution to the concurrent probabilistic system $\mathcal{S}^{Target}_{T}$ (recall that concurrent probabilistic systems are essentially equivalent to the Markov decision process models of Courcoubetis and Yannakakis, and de Alfaro), where the set of destination states (see [dA97a]) is equal to *Reached*. Let $\Pr(\mathsf{close}(\mathsf{time\_succ}(\bar{v}, \mathbf{0}), c)$ be the maximal reachability probability value computed for the symbolic state $\mathsf{close}(\mathsf{time\_succ}(\bar{v}, \mathbf{0}), c)$. Then, for a given probabilistic timed automaton T and a probabilistic real-time reachability problem $(\mathit{Target}, \sqsupseteq, \lambda)$, the answer to the problem is:

- "Maybe", if $\mathit{Reached} \neq \emptyset$ and $\Pr(\mathsf{close}(\mathsf{time\_succ}(\bar{v}, \mathbf{0}), c)) \sqsupseteq \lambda$, and

- "NO", otherwise.

Recall that invariance properties can be stated in terms of reachability properties, and that if the answer to the reachability property is "YES" ("NO") then the answer to the invariant property is "NO" ("YES", respectively). The introduction of the possibility of a "MAYBE" result to the probabilistic real-time reachability problem has the following implications for the verification of invariance properties: if the answer to the reachability problem obtained from the invariance problem is "MAYBE", then the answer to the invariance problem is also "MAYBE"; however, if "NO" is returned, then the answer to the invariance problem is "YES". This leads us to conclude that our forward reachability method may be particularly appropriate for establishing the satisfaction of probabilistic, real-time invariance properties.

6.4 Example of probabilistic real-time reachability

Consider again the probabilistic timed automaton T_{comm} of Figure 4.5. Say we are interested in the untimed, probabilistic reachability property "with probability 0.9999 or greater, it is possible to correctly deliver a data packet" (we consider an untimed property for simplicity). This requirement can be specified in terms of the probabilistic real-time reachability problem $(\{\mathsf{ri}\}, \geq, 0.9999)$, with the computation of the algorithm **ForwardReachability** taking the form given in Figure 6.3. A typical iteration of the algorithm consists of the following steps:

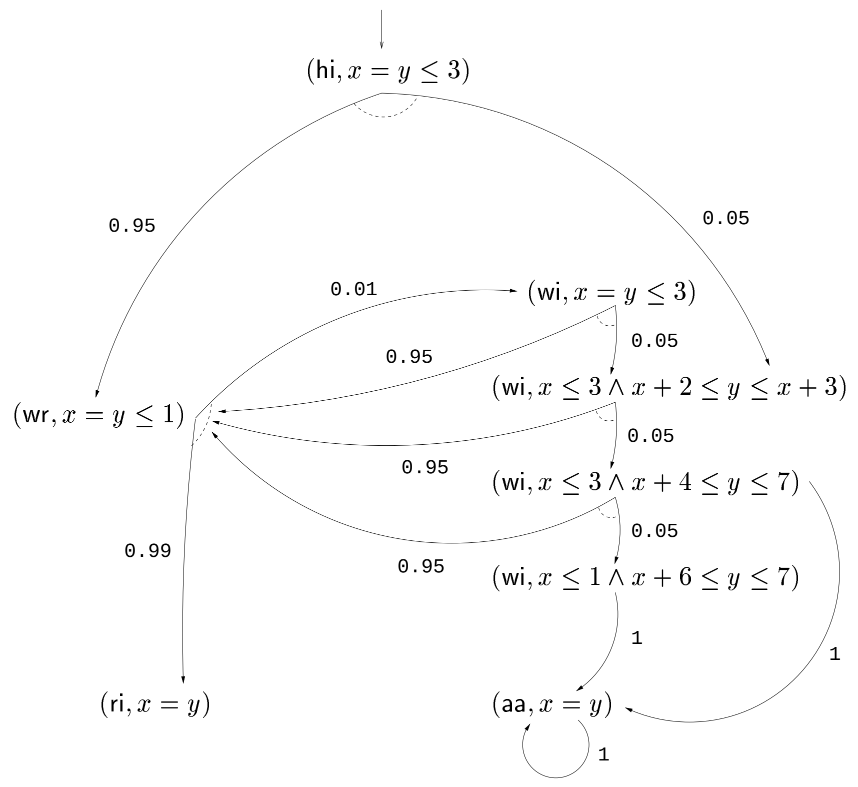
1. a symbolic state (v, ζ) is taken from *Fringe*;
2. second, for all edges that are enabled in (v, ζ), the discrete successor is generated;
3. third, the time successor of each discrete successor computed in the previous step are generated;
4. finally, the sets *Fringe* and *SymbStates* are adjusted, so that *Fringe* now contains the newly generated symbolic states (provided that they do not already exist in *Fringe* or *SymbStates*), and (v, ζ) is added to *SymbStates*.

Note that the second and third steps relate to the application of the post operation, and that the $c_{\max}(T)$-closure of the time successors is implicit (as $c_{\max}(T) = 7$, we take the 7-closure of the time successors). The notation $v \to v'$, for locations $v, v' \in V$, refers to the edge from location v to v' A special case is iteration 4, in which the node of the symbolic state $(\mathsf{ri}, x = y \leq 1)$ is found to be in the target set *Target*, and therefore $(\mathsf{ri}, x = y)$ is added to *Reached*.

The resulting forward reachability graph $\mathcal{S}^{\{\mathsf{ri}\}}_{T_{\mathrm{comm}}}$ is shown in figure Figure 6.4. Observe that the only nondeterministic choice made in this concurrent probabilistic system is featured in the symbolic state $(\mathsf{wi}, x \leq 3 \wedge x + 4 \leq y \leq 7)$, and that the

Initially: $\mathit{Fringe} = \{(\mathsf{hi}, x = y \leq 3)\}$, $\mathit{SymbStates} = \emptyset$			
Iteration 1: take $(\mathsf{hi}, x = y \leq 3)$ from Fringe			
disc_succ	$\mathsf{hi} \to \mathsf{wr} : (\mathsf{wr}, x = 0 \wedge y = 0)$	$\mathsf{hi} \to \mathsf{wi} : (\mathsf{wi}, x = 0 \wedge 2 \leq y \leq 3)$	
time_succ	$\mathsf{hi} \to \mathsf{wr} : (\mathsf{wr}, x = y \leq 1)$	$\mathsf{hi} \to \mathsf{wi} : (\mathsf{wi}, x \leq 3 \wedge x + 2 \leq y \leq x + 3)$	
$\mathit{Fringe} = \{(\mathsf{wr}, x = y \leq 1), (\mathsf{wi}, x \leq 3 \wedge x + 2 \leq y \leq x + 3)\}$, $\mathit{SymbStates} = \{(\mathsf{hi}, x = y \leq 3)\}$			
Iteration 2: take $(\mathsf{wr}, x = y \leq 1)$ from Fringe			
disc_succ	$\mathsf{wr} \to \mathsf{ri} : (\mathsf{ri}, x = y \leq 1)$	$\mathsf{wr} \to \mathsf{wi} : (\mathsf{wi}, x = y \leq 1)$	
time_succ	$\mathsf{wr} \to \mathsf{ri} : (\mathsf{ri}, x = y)$	$\mathsf{wr} \to \mathsf{wi} : (\mathsf{wi}, x = y \leq 3)$	
$\mathit{Fringe} = \{(\mathsf{wi}, x \leq 3 \wedge x + 2 \leq y \leq x + 3), (\mathsf{ri}, x = y), (\mathsf{wi}, x = y \leq 3)\}$, $\mathit{SymbStates} = \{(\mathsf{hi}, x = y \leq 3), (\mathsf{wr}, x = y \leq 1)\}$			
Iteration 3: take $(\mathsf{wi}, x \leq 3 \wedge x + 2 \leq y \leq x + 3)$ from Fringe			
disc_succ	$\mathsf{wi} \to \mathsf{wr} : (\mathsf{wr}, x = 0 \wedge y = 0)$	$\mathsf{wi} \to \mathsf{wi} : (\mathsf{wi}, x = 0 \wedge 4 \leq y \leq 6)$	
time_succ	$\mathsf{wi} \to \mathsf{wr} : (\mathsf{wr}, x = y \leq 1)$	$\mathsf{wi} \to \mathsf{wi} : (\mathsf{wi}, x \leq 3 \wedge x + 4 \leq y \leq 7)$	
$\mathit{Fringe} = \{(\mathsf{ri}, x = y), (\mathsf{wi}, x = y \leq 3), (\mathsf{wi}, x \leq 3 \wedge x + 4 \leq y \leq 7)\}$, $\mathit{SymbStates} = \{(\mathsf{hi}, x = y \leq 3), (\mathsf{wr}, x = y \leq 1), (\mathsf{wi}, x \leq 3 \wedge x + 2 \leq y \leq x + 3)\}$			
Iteration 4: take $(\mathsf{ri}, x = y)$ from Fringe			
Add $(\mathsf{ri}, x = y)$ to Reached			
$\mathit{Fringe} = \{(\mathsf{wi}, x = y \leq 3), (\mathsf{wi}, x \leq 3 \wedge x + 4 \leq y \leq 7)\}$, $\mathit{SymbStates} = \{(\mathsf{hi}, x = y \leq 3), (\mathsf{wr}, x = y \leq 1), (\mathsf{wi}, x \leq 3 \wedge x + 2 \leq y \leq x + 3), (\mathsf{ri}, x = y)\}$			
Iteration 5: take $(\mathsf{wi}, x = y \leq 3)$ from Fringe			
disc_succ	$\mathsf{wi} \to \mathsf{wr} : (\mathsf{wr}, x = 0 \wedge y = 0)$	$\mathsf{wi} \to \mathsf{wi} : (\mathsf{wi}, x = 0 \wedge 2 \leq y \leq 3)$	
time_succ	$\mathsf{wi} \to \mathsf{wr} : (\mathsf{wr}, x = y \leq 1)$	$\mathsf{wi} \to \mathsf{wi} : (\mathsf{wi}, x \leq 3 \wedge x + 2 \leq y \leq x + 3)$	
$\mathit{Fringe} = \{(\mathsf{wi}, x \leq 3 \wedge x + 4 \leq y \leq 7)\}$, $\mathit{SymbStates} = \{(\mathsf{hi}, x = y \leq 3), (\mathsf{wr}, x = y \leq 1), (\mathsf{wi}, x \leq 3 \wedge x + 2 \leq y \leq x + 3), (\mathsf{ri}, x = y), (\mathsf{wi}, x = y \leq 3)\}$			
Iteration 6: take $(\mathsf{wi}, x \leq 3 \wedge x + 4 \leq y \leq 7)$ from Fringe			
disc_succ	$\mathsf{wi} \to \mathsf{wr} : (\mathsf{wr}, x = 0 \wedge y = 0)$	$\mathsf{wi} \to \mathsf{wi} : (\mathsf{wi}, x = 0 \wedge 6 \leq y \leq 7)$	$\mathsf{wi} \to \mathsf{aa} : (\mathsf{aa}, x = 0 \wedge y = 0)$
time_succ	$\mathsf{wi} \to \mathsf{wr} : (\mathsf{wr}, x = y \leq 1)$	$\mathsf{wi} \to \mathsf{wi} : (\mathsf{wi}, x \leq 1 \wedge x + 6 \leq y \leq 7)$	$\mathsf{wi} \to \mathsf{aa} : (\mathsf{aa}, x = y)$
$\mathit{Fringe} = \{(\mathsf{wi}, x \leq 1 \wedge x + 6 \leq y \leq 7), (\mathsf{aa}, x = y)\}$, $\mathit{SymbStates} = \{(\mathsf{hi}, x = y \leq 3), (\mathsf{wr}, x = y \leq 1), (\mathsf{wi}, x \leq 3 \wedge x + 2 \leq y \leq x + 3), (\mathsf{ri}, x = y), (\mathsf{wi}, x = y \leq 3),$ $(\mathsf{wi}, x \leq 3 \wedge x + 4 \leq y \leq 7)\}$			
Iteration 7: take $(\mathsf{wi}, x \leq 1 \wedge x + 6 \leq y \leq 7)$ from Fringe			
disc_succ	$\mathsf{wi} \to \mathsf{aa} : (\mathsf{aa}, x = 0 \wedge y = 0)$		
time_succ	$\mathsf{wi} \to \mathsf{aa} : (\mathsf{aa}, x = y)$		
$\mathit{Fringe} = \{(\mathsf{aa}, x = y)\}$, $\mathit{SymbStates} = \{(\mathsf{hi}, x = y \leq 3), (\mathsf{wr}, x = y \leq 1), (\mathsf{wi}, x \leq 3 \wedge x + 2 \leq y \leq x + 3), (\mathsf{ri}, x = y), (\mathsf{wi}, x = y \leq 3),$ $(\mathsf{wi}, x \leq 3 \wedge x + 4 \leq y \leq 7), (\mathsf{wi}, x \leq 1 \wedge x + 6 \leq y \leq 7)\}$			
Iteration 8: take $(\mathsf{aa}, x = y)$ from Fringe			
disc_succ	$\mathsf{aa} \to \mathsf{aa} : (\mathsf{aa}, x = 0 \wedge y = 0)$		
time_succ	$\mathsf{aa} \to \mathsf{aa} : (\mathsf{aa}, x = y)$		
$\mathit{Fringe} = \emptyset$, $\mathit{SymbStates} = \{(\mathsf{hi}, x = y \leq 3), (\mathsf{wr}, x = y \leq 1), (\mathsf{wi}, x \leq 3 \wedge x + 2 \leq y \leq x + 3), (\mathsf{ri}, x = y), (\mathsf{wi}, x = y \leq 3),$ $(\mathsf{wi}, x \leq 3 \wedge x + 4 \leq y \leq 7), (\mathsf{wi}, x \leq 1 \wedge x + 6 \leq y \leq 7), (\mathsf{aa}, x = y)\}$			
$\mathit{Fringe} = \emptyset$, therefore terminate			

Figure 6.3: Computations of **ForwardReachability** on T_{comm}.



$(\mathsf{hi}, x = y \leq 3)$
0.95
0.05
0.01
$(\mathsf{wi}, x = y \leq 3)$
0.05
0.95
$(\mathsf{wi}, x \leq 3 \wedge x + 2 \leq y \leq x + 3)$
$(\mathsf{wr}, x = y \leq 1)$
0.05
0.95
$(\mathsf{wi}, x \leq 3 \wedge x + 4 \leq y \leq 7)$
0.05
0.99
0.95
$(\mathsf{wi}, x \leq 1 \wedge x + 6 \leq y \leq 7)$
1
1
$(\mathsf{ri}, x = y)$
$(\mathsf{aa}, x = y)$
1

Figure 6.4: The forward reachability graph $\mathcal{S}^{\{\mathsf{ri}\}}_{T_{comm}}$.

single destination state of $\mathcal{S}^{\{\mathsf{ri}\}}_{T_{\mathrm{comm}}}$ is the symbolic state $(\mathsf{ri}, x = y)$ in *Reached*. After application of a linear programming problem, we find that the maximal reachability probability value of reaching $(\mathsf{ri}, x = y)$ from the initial state $(\mathsf{hi}, x = 0 \wedge y = 0)$ is 0.9998737375. As this value is less than 0.9999, we conclude that the answer to the reachability problem, "with probability 0.9999 or greater, it is possible to correctly deliver a data packet", is "No." Observe that the region graph of T_{comm} and the PTCTL property $[\exists\Diamond\mathsf{ri}]_{\geq 0.9999}$, which is equivalent to the reachability property $(\{\mathsf{ri}\}, \geq, 0.9999)$, consists of 101 reachable states, compared to the 8 states constructed by the forward reachability method of Figure 6.1.

Note that it is straightforward to verify T_{comm} against a time bounded reachability property such as "with probability 0.975 or greater, it is possible to correctly deliver a data packet within 5 time units". As explained in Section 6.2, all that is required is to augment T_{comm} with an additional clock z, an extra location v', and an additional distribution in the node ri, which is enabled only when $z \leq 5$ and leads to the new target location v' with probability 1. We then perform reachability analysis on the new probabilistic timed automaton in a similar way to that described above. It also follows that verification of invariance properties of T_{comm}, such as "with probability

0.875 or greater, the system never aborts" is possible.

6.5 Issues of progress for reachability

We now introduce a notion of divergence to the reachability problem, in order to reflect the requirements of discrete and time progress of Section 4.4. A number of alternatives are available to us, including disregarding issues of divergence, or simply adopting the ideas of Section 4.4 wholesale. For practical reasons, we prefer an intermediate approach to the either alternative, which is characterised in the following way: for a target set $\mathit{Target} \subseteq V$, a path of the concurrent probabilistic system $\mathcal{S}_T$ is *Target*-divergent if and only if it either reaches *Target* or it is divergent in the sense of Definition 4.4.3. Then, an adversary of $\mathcal{S}_T$ is *Target*-divergent if the probability measure over its set of *Target*-divergent paths is 1.

We present a justification for this choice. One of the benefits enjoyed by forward reachability algorithms for non-probabilistic timed automata [DOTY96, LPY97a, DT98] is their on-the-fly nature; that is, they can terminate as soon as a reachable state set that intersects the target set is computed with a "Yes" answer. This can, in part, explain the relative success of the tools Kronos and Uppaal, which implement these algorithms. As noted by Tripakis [Tri98], a path which reaches the target state may *not* necessarily be divergent; for example, a target set may correspond to a set of states from which, after being reached, subsequent discrete or time progress is impossible. However, to obtain information concerning the divergence of such paths, infinite runs through symbolic states (corresponding to cycles of symbolic states) should be constructed, potentially necessitating additional computation. We prefer to take an intermediate approach for the case of probabilistic real-time reachability, in the sense that the algorithm **ForwardReachability** does not explore any successors of symbolic states that correspond to a target location, suggesting that, as in the non-probabilistic context, the concept of divergence should be abandoned when performing reachability analysis. However, we postulate that a meaningful notion of probabilistic divergence can nevertheless be applied in our context.

For an example of this notion, consider the following fragment of a probabilistic timed automaton, T_{div}, shown in Figure 6.5, and the target set $\{v_2\}$. Then, for all adversaries of $\mathcal{S}_{T_{\mathrm{div}}}$ that take the illustrated distribution in $\bar{v}$, there is a probability of $\frac{1}{2}$ of not being allowed to let time diverge; this corresponds to the transition from the location $\bar{v}$ to v_1, and the fact that all states of the location v_1 are timelocks. However, if the reachability problem was of the form $(\{v_2\}, \geq, 0.5)$, then we conclude that the answer to the problem is "Yes" (regardless of the remainder of the locations and transitions of T_{div}, such as the "timelocking" location v_1). We regard such an eventuality as being undesirable, and would prefer to disregard such adversaries from our analysis.

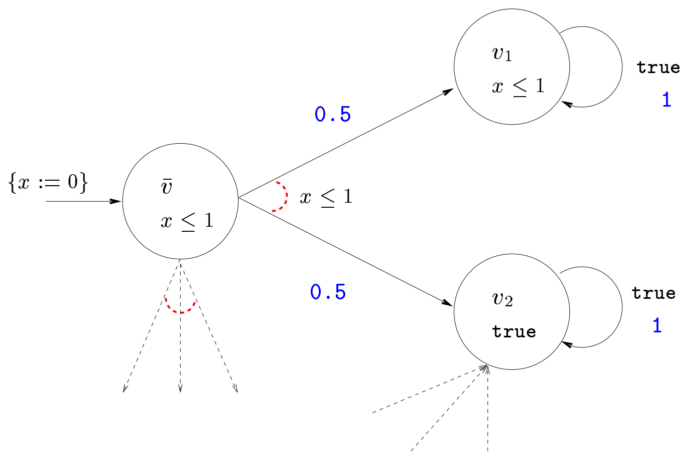


Figure 6.5: A fragment of the probabilistic timed automaton T_{div}.

This justifies the following definition of *Target*-divergent paths and adversaries, both of the concurrent probabilistic system $\mathcal{S}_T$ of T and of the forward reachability graph $\mathcal{S}_T^{\mathit{Target}}$.

Definition 6.5.1 **(*Target*-divergent paths of $\mathcal{S}_T$)** *For the probabilistic timed automaton T, and a target set $\mathit{Target} \subseteq V$ of locations, a path $\omega \in \mathit{Path}_{\mathit{ful}}$ of the concurrent probabilistic system $\mathcal{S}_T$ of T is Target-*divergent *if it is either divergent or there exists $i \in \mathbb{N}$ such that* $\mathsf{discrete}(\omega(i)) \in \mathit{Target}$.

Definition 6.5.2 **(*Target*-divergent adversaries of $\mathcal{S}_T$)** *For the probabilistic timed automaton T with the concurrent probabilistic system $\mathcal{S}_T$, and a target set $\mathit{Target} \subseteq V$ of locations, an adversary $A \in \mathit{Adv}$ of $\mathcal{S}_T$ is Target-divergent if and only if*

$$\mathit{Prob}^A(\{\omega \in \mathit{Path}_{\mathit{ful}}^A \mid \omega \text{ is } \mathit{Target}\text{-divergent}\}) = 1.$$

Let $\mathit{Div}_{\mathit{Target}}$ be the set of all Target-divergent adversaries of $\mathcal{S}_T$.

Observe that the notion of *Target*-divergence for $\mathcal{S}_T$ is weaker than that of divergence, both for the case of paths and adversaries. That is, $\mathit{Div} \subseteq \mathit{Div}_{\mathit{Target}} \subseteq \mathit{Adv}$.

We now consider how *Target*-divergence can be defined on paths and adversaries of the forward reachability graph $\mathcal{S}_T^{\mathit{Target}}$. Our approach to the definition of divergence of paths of $\mathcal{S}_T^{\mathit{Target}}$ is mainly inspired by the non-probabilistic precedent of [Tri98]. The notion of "unboundedness" used in Section 5.4 is used to define the predicate $\mathsf{unbounded}$, in order to express the unboundedness of a clock in a certain zone. Given a clock $x_i \in \mathcal{X}$ and a zone $\zeta \in \mathsf{Zone}(\mathcal{X})$, we define:

$$\mathsf{unbounded}(x_i, \zeta) \overset{\mathit{def}}{=} \begin{cases} \texttt{true} & \text{if } \forall t \in \mathbb{R}_{\geq 0} \,.\, \exists \mathbf{a} \in \zeta \,.\, \mathbf{a}_i > t \\ \texttt{false} & \text{otherwise.} \end{cases}$$

For a path of $\mathcal{S}_T^{Target}$ to be divergent, we require that each clock of T either becomes arbitrarily large or is reset infinitely often with some nonzero probability. The predicate $\mathsf{unbounded}$ is used to express the first requirement in a straightforward manner, while, for the second requirement, it suffices for a distribution for which a probabilistic choice corresponds to a clock being reset is chosen infinitely often (as opposed to the more intuitive, but stronger requirement which simply states that the clock must be reset infinitely often along the path). To support this claim, consider the following informal argument. Take an infinite path $\pi \in Path_{ful}(\langle \bar{s} \rangle)$ of the forward reachability graph $\mathcal{S}_T^{Target}$, and say that this path features infinitely many transitions of the form $(v, \zeta) \xrightarrow{\theta_p, \hat{p}} (v', \zeta')$ for which there exists an edge $(v, v', X, p) \in \mathsf{edge\_support}(p)$ such that $x \in X$, and $\mathsf{post}[(v, v', X, p), c](v, \zeta) = (v', \zeta')$. Note that there can be another edge $(v, v', X', p) \in \mathsf{edge\_support}(p)$ such that $x \notin X'$ which is such that $\mathsf{post}[(v, v', X', p), c](v, \zeta) = (v', \zeta')$ also. However, the probability of a path in which the former edge (which is such that $x \in X$) is chosen *finitely often* is 0, by the weak law of large numbers (see [Bil86]). Therefore, it suffices to characterise paths of the forward reachability graph which reset a certain clock $x \in \mathcal{X}$ infinitely often by the transitions between particular symbolic states made according to particular distributions. We can now formalise this concept.

Definition 6.5.3 *Let T be probabilistic timed automaton, $Target \subseteq V$ be a set of target locations, and let $\mathcal{S}_T^{Target}$ be the forward reachability graph. Given a clock $x \in \mathcal{X}$, we say that the transition $(v, \zeta) \xrightarrow{\theta_p, \hat{p}} (v', \zeta')$ of $\mathcal{S}_T^{Target}$ is:*

- unbounded in x *if and only if:*

$$\mathsf{unbounded}(x, \zeta) \wedge \mathsf{unbounded}(x, pre_v(p)) = \mathtt{true}\,;$$

- a reset of x *if there exists an edge $(v, v', X, p) \in \mathsf{edge\_support}(p)$ for which $x \in X$ and $(v', \zeta') = \mathsf{post}[(v, v', X, p), c](v, \zeta)$; and*

- x-zero *if $pre_v(p)$, when written in syntactic form, includes the conjunct $x = 0$.*

We now use this characterisation of transitions to express the fact that a path of the forward reachability graph is divergent if and only if, for each clock $x \in \mathcal{X}$, the transitions of the paths are unbounded in x after a certain point, or the clock $x \in \mathcal{X}$ is reset infinitely often with positive probability. Because of the presence of reachable states, we only need to consider paths which do not visit a location in *Target*. We then define divergent adversaries on the forward reachability graph in a manner similar to that of Definition 6.5.2.

Definition 6.5.4 (Divergent paths of $\mathcal{S}_T^{Target}$) *For the probabilistic timed automaton T, the set of target locations $Target \subseteq V$, and the forward reachability graph $\mathcal{S}_T^{Target}$, a path $\pi \in Path_{ful}(\langle \bar{s} \rangle)$, which is of the form $\langle \bar{s} \rangle \xrightarrow{\sigma_0, \hat{p}_0} (v_1, \zeta_1) \xrightarrow{\sigma_1, \hat{p}_1} \ldots$ is:*

possibly progresive *if for each clock $x \in \mathcal{X}$, either of the following holds:*

- *for every $i \in \mathbb{N}$, there exists $j > i$ such that the transition $\pi(j) \xrightarrow{\sigma_j,\hat{p}_j} \pi(j+1)$ is a reset of x, or*
- *there exists $i \in \mathbb{N}$ such that, for all $j > i$, the transition $\pi(j) \xrightarrow{\sigma_j,\hat{p}_j} \pi(j+1)$ is unbounded in x;*

zero *if there exists a clock $x \in \mathcal{X}$ such that, there exists $i \in \mathbb{N}$ such that, for all $j > i$, the transition $\pi(j) \xrightarrow{\sigma_j,\hat{p}_j} \pi(j+1)$ is x-zero;*

divergent *if it is possibly progressive and not zero.*

Definition 6.5.5 (*Target*-divergent paths of $\mathcal{S}_T^{Target}$) *Let T be a probabilistic timed automaton, $Target \subseteq V$ be a set of target locations, and $\mathcal{S}_T^{Target}$ be the resulting forward reachability graph $\mathcal{S}_T^{Target}$. A path $\pi \in Path_{ful}(\langle \bar{s} \rangle)$ of $\mathcal{S}_T^{Target}$ is Target-*divergent *if*

- *it is divergent, or*
- *there exists $i \in \mathbb{N}$ such that $\mathsf{discrete}(\pi(i)) \in Target$.*

Note that the second point in Definition 6.5.5 is equivalent to the statement, "if there exists $i \in \mathbb{N}$ such that $\mathsf{discrete}(\pi(j)) \in Target$ for all $j > i$". The reason for this is that, if a path of the forward reachability graph $\mathcal{S}_T^{Target}$ reaches a target location in *Target*, then, by the construction of $\mathcal{S}_T^{Target}$, the path will loop in the same symbolic state for the remainder of the path.

Definition 6.5.6 (*Target*-divergent adversaries of $\mathcal{S}_T^{Target}$) *For the probabilistic timed automaton T, the set of target locations $Target \subseteq V$, and the forward reachability graph $\mathcal{S}_T^{Target}$, an adversary $B \in \langle Adv \rangle$ of the forward reachability graph $\mathcal{S}_T^{Target}$ is Target-*divergent *if and only if*

$$Prob(\{\pi \in Path^B_{ful}(\langle \bar{s} \rangle) \mid \pi \text{ is } Target\text{-divergent}\}) = 1.$$

Let $\langle Div_{Target} \rangle$ be the set of all Target-divergent adversaries of $\mathcal{S}_T^{Target}$.

6.6 Proof of correctness

We now show the correctness of the algorithm **ForwardReachability** of Figure 6.1, together with the construction process of the forward reachability graph of Definition 6.3.3. We have given an example which shows that our forward reachability procedure will compute a maximal reachability probability on the forward reachability graph that may be greater than the actual maximal probability on the probabilistic timed automaton T. However, we now show that the computed probability will not be *less* than the actual probability by proving that, for every adversary of the concurrent probabilistic system $\mathcal{S}_T$ of T, there exists a sufficiently "similar" adversary of the forward reachability graph $\mathcal{S}_T^{\mathit{Target}}$, where the notion of "similarity" means that the probability of reaching the target set of locations *Target* is the same for both adversaries. The proof proceeds in two steps: first, properties of single transitions of the forward reachability graph $\mathcal{S}_T^{\mathit{Target}}$ and the concurrent probabilistic system $\mathcal{S}_T$ of T are studied, providing the basis of the second step, which concerns the paths and adversaries of $\mathcal{S}_T$ and $\mathcal{S}_T^{\mathit{Target}}$.

6.6.1 Properties of single-step transitions

We commence by exploring the way in which a single transition of the concurrent probabilistic system $\mathcal{S}_T$ of the probabilistic timed automaton T may be mimicked by a transition of the forward reachability graph $\mathcal{S}_T^{\mathit{Target}}$, such that both transitions are derived from the same original distribution p of T (as encoded in the event θ_p). Our concerns are twofold: first we show that, for any state $(v, \mathbf{a})$ of $\mathcal{S}_T$ contained within a symbolic state (v, ζ) of $\mathcal{S}_T^{\mathit{Target}}$, any compound transition from $(v, \mathbf{a})$, comprising a time transition followed by a discrete transition, can be mimicked by a transition from (v, ζ) with the same discrete transition event θ_p. Then we show that, for any target state $(v', \mathbf{b})$ of the transition from $(v, \mathbf{a})$, the set of clock resets which can be used to obtain $(v', \mathbf{b})$ can be used to obtain a set of successor symbolic states of (v, ζ), *each element of which contains* $(v', \mathbf{b})$.

Existence of transitions

First, we show that the existence of a transition from a state $(v, \mathbf{a})$ of $\mathcal{S}_T$ implies the existence of a transition with the same event θ_p from any symbolic state of $\mathcal{S}_T^{\mathit{Target}}$ which contains $(v, \mathbf{a})$, and the target of the latter transition contains the target of the former. In the following, we often refer to the consecutive time and discrete transitions denoted by $(v, \mathbf{a}) \xrightarrow{\delta} \xrightarrow{\theta_p, \tilde{p}} (v', \mathbf{b})$ as a single transition.

Lemma 6.6.1 *Consider the probabilistic timed automaton T, the target set Target $\subseteq V$, and the resulting forward reachability graph $\mathcal{S}_T^{\mathit{Target}}$. Then, for any symbolic state $(v, \zeta) \in S_T^{\mathit{Target}}$ such that $v \notin$ Target, state $(v, \mathbf{a}) \in S_T$ such that $(v, \mathbf{a}) \in (v, \zeta)$, and transition $(v, \mathbf{a}) \xrightarrow{\delta} \xrightarrow{\theta_p, \tilde{p}} (v', \mathbf{b})$ of $\mathcal{S}_T$, there exists a transition $(v, \zeta) \xrightarrow{\theta_p, \hat{p}} (v', \zeta')$ of $\mathcal{S}_T^{\mathit{Target}}$, such that $(v', \mathbf{b}) \in (v', \zeta')$.*

Proof. The correctness of this lemma follows from the definitions of post and of the forward reachability graph $\mathcal{S}_T^{\mathit{Target}}$ (Definition 6.3.2 and Definition 6.3.3, respectively). Given the distribution $p \in \mathit{prob}(v)$ that appears in the subscript of the event θ_p, we observe that there must exist a clock reset set $X \subseteq \mathcal{X}$ such that both $\mathbf{b} = \mathbf{a}[X := 0]$ and $p(v', X) > 0$. Furthermore, from $p(v', X) > 0$, there exists an edge $e = (v, v', X, p)$ in the set of edges of T.

Because the transition $(v, \mathbf{a}) \xrightarrow{\delta, \theta_p, \tilde{p}} (v', \mathbf{b})$ and the fact that the distribution p is the subscript of the θ_p encode the information of letting δ time units elapse from the state $(v, \mathbf{a})$, and then choosing the distribution p, we conclude that $\mathbf{a} + \delta \in \mathit{pre}_v(p)$.

From the definition of post (and the fact that the initial symbolic state of the forward reachability graph is $\mathsf{close}(\mathsf{time\_succ}(\bar{v}, \mathbf{0}), c))$, all of the time successors of any state in (v, ζ), and all of the states that are c-equivalent to such time successors, are also in (v, ζ). From this fact, we conclude that $\mathbf{a} + \delta \in \zeta$. Then, putting $\mathbf{a} + \delta \in \zeta$ and $\mathbf{a} + \delta \in \mathit{pre}_v(p)$ together, we obtain $\zeta \cap \mathit{pre}_v(p) \neq \emptyset$.

Then, by the assumption of tightened pre-conditions of T (see Definition 4.2.2), we have $((\zeta \cap \mathit{pre}_v(p))[X := 0]) \cap \mathit{inv}(v') \neq \emptyset$. Hence, $\mathsf{disc\_succ}(e, (v, \zeta)) = (v', \zeta'')$, where $\zeta'' \neq \emptyset$. As $\mathbf{a} + \delta \in \mathit{pre}_v(p)$, and $\mathbf{b} = (\mathbf{a} + \delta)[X := 0]$, it must be the case that $\mathbf{b} \in \zeta''$, and therefore $(v', \mathbf{b}) \in (v', \zeta'')$.

Next, because $\mathsf{close}(\mathsf{time\_succ}(v', \zeta''), c) \supseteq (v', \zeta'')$, we have $(v', \mathbf{b}) \in \mathsf{post}[e, c](v, \zeta)$. Then we let $(v', \zeta') = \mathsf{post}[e, c](v, \zeta)$ and $(v', \mathbf{b}) \in (v', \zeta')$ as required. □

Targets of transitions

We now proceed to the main lemma of this section. Intuitively, we highlight the correspondence between transitions of the concurrent probabilistic system $\mathcal{S}_T$ and the forward reachability graph $\mathcal{S}_T^{\mathit{Target}}$ by establishing the following facts. Consider the state $(v, \mathbf{a})$, the symbolic state (v, ζ) such that $(v, \mathbf{a}) \in (v, \zeta)$, and the transitions denoted by $(v, \mathbf{a}) \xrightarrow{\delta, \theta_p, \tilde{p}}$ and $(v, \zeta) \xrightarrow{\theta_p, \hat{p}}$ (that is, two transitions which are derived from the same distribution $p \in \mathit{prob}(v)$ of the probabilistic timed automaton T). Then:

- every state of $\mathcal{S}_T$ which is in the support of $\tilde{p}$ is contained in a symbolic state in the support of $\hat{p}$; conversely, every symbolic state of $\mathcal{S}_T^{\mathit{Target}}$ in the support of $\hat{p}$ contains a state in the support of $\tilde{p}$;

- furthermore, with the intuition that the probability $\tilde{p}(v', \mathbf{b})$ is equal to the sum of probabilities assigned by the distribution p to edges whose target is v' and which reset clocks in such a way as to result in the valuation $\mathbf{b}$, then traversing each of these edges from the symbolic state (v, ζ) will result in a target symbolic state (v', ζ') (not necessarily the same for each edge) which contains $(v', \mathbf{b})$.

First, consider the following facts concerning the effects of clock resets on valuations and zones. Given a valuation $\mathbf{a} \in \mathbb{R}^n_{\geq 0}$, a particular valuation $\mathbf{b} \in \mathbb{R}^n_{\geq 0}$ may be obtained via more than one clock reset. For example, if $\mathcal{X} = \{x, y\}$, and $\mathbf{a}(x) = \mathbf{a}(y) = 0$, then $\mathbf{a}[\{x\} := 0] = \mathbf{a}[\{y\} := 0] = \mathbf{a}[\{x, y\} := 0]$. However, note that the application of these reset sets to a given zone ζ may result in *different* zones. Consider the zone ζ given by the constraint $x \leq y$. Then $\mathbf{a} \in \zeta$, yet $\zeta[\{x\} := 0]$ is defined by $0 = x \leq y$, whereas $\zeta[\{y\} := 0]$ is defined by $0 = y \leq x$, and $\zeta[\{x, y\} := 0]$ is defined by $x = y = 0$.

These concepts can be described formally in the following way.

Lemma 6.6.2 *Let $\mathcal{X}$ be a set of clocks, $\zeta \in \mathsf{Zone}(\mathcal{X})$ be a zone, and $X_1, X_2 \subseteq \mathcal{X}$ be sets of clocks such that $\zeta[X_1 := 0] = \zeta[X_2 := 0]$. If $\mathbf{a} \in \zeta$, then $\mathbf{a}[X_1 := 0] = \mathbf{a}[X_2 := 0]$.*

The lemma states that, if the application of two different resets to a zone ζ results in the same zone ζ', then the application of these resets to a valuation $\mathbf{a}$ contained in ζ must result in the same valuation $\mathbf{b}$. However, the converse of Lemma 6.6.2 (that, if $\mathbf{a}[X_1 := 0] = \mathbf{a}[X_2 := 0]$ and $\mathbf{a} \in \zeta$, then $\zeta[X_1 := 0] = \zeta[X_2 := 0]$) is *not* necessarily true. Therefore, although multiple reset sets may result in the same valuation, these sets may result in different zones, although it is clear that the valuation will be contained within *all* such zones. Consider again the above example. Although $\zeta[\{x\} := 0] \neq \zeta[\{y\} := 0] \neq \zeta[\{x, y\} := 0]$, it is the case that $\mathbf{b} = \mathbf{a}[\{x\} := 0] = \mathbf{a}[\{y\} := 0] = \mathbf{a}[\{x, y\} := 0]$ is such that $\mathbf{b} \in \zeta[\{x\} := 0]$, $\mathbf{b} \in \zeta[\{y\} := 0]$ and $\mathbf{b} \in \zeta[\{x, y\} := 0]$.

We now introduce three functions in order to reason about the relationship between valuations and zones. Let $\mathcal{X}$ be any set of clocks.

- $\mathsf{SResets} : \mathbb{R}^n_{\geq 0} \times \mathbb{R}^n_{\geq 0} \to 2^{2^{\mathcal{X}}}$ is the function such that, for any $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n_{\geq 0}$,
$$\mathsf{SResets}(\mathbf{a}, \mathbf{b}) \overset{\text{def}}{=} \{X \subseteq \mathcal{X} \mid \mathbf{a}[X := 0] = \mathbf{b}\} .$$
- Similarly, $\mathsf{ZResets} : \mathsf{Zone}(\mathcal{X}) \times \mathsf{Zone}(\mathcal{X}) \to 2^{2^{\mathcal{X}}}$ is the function such that, for any $\zeta, \zeta' \in \mathsf{Zone}(\mathcal{X})$,
$$\mathsf{ZResets}(\zeta, \zeta') \overset{\text{def}}{=} \{X \subseteq \mathcal{X} \mid \zeta[X := 0] = \zeta'\} .$$
- Finally, $\mathsf{NewZones} : \mathsf{Zone}(\mathcal{X}) \times 2^{2^{\mathcal{X}}} \to 2^{\mathsf{Zone}(\mathcal{X})}$ is the function such that, for any $\zeta \in \mathsf{Zone}(\mathcal{X})$ and $\mathcal{Y} \subseteq 2^{\mathcal{X}}$,
$$\mathsf{NewZones}(\zeta, \mathcal{Y}) \overset{\text{def}}{=} \{\zeta' \in \mathsf{Zone}(\mathcal{X}) \mid \zeta' = \zeta[X := 0] \text{ for some } X \in \mathcal{Y}\} .$$

Note that we always define the first two functions with respect to a particular set of clocks, and therefore we should write $\mathsf{SResets}_{\mathcal{X}}$ and $\mathsf{ZResets}_{\mathcal{X}}$. However, we omit such subscripts to ease notation, and always make it clear which clock set $\mathsf{SResets}$ and $\mathsf{ZResets}$ are defined with respect to.

The intuition underlying these functions is as follows. The set $\mathsf{SResets}(\mathbf{a}, \mathbf{b})$ comprises those clock resets which, if applied to the valuation $\mathbf{a}$, result in the valuation $\mathbf{b}$. The set $\mathsf{ZResets}(\zeta, \zeta')$ contains those clock resets which, if applied to the zone ζ, result in the zone ζ' (note that $\mathsf{SResets}(\mathbf{a},\mathbf{b})$ and $\mathsf{ZResets}(\zeta,\zeta')$ may contain the empty clock reset $\emptyset$). Finally, $\mathsf{NewZones}(\zeta,\mathcal{Y})$ is the set of zones which can be obtained by the application of clock resets in $\mathcal{Y}$ to ζ.

The following lemma follows immediately from Lemma 6.6.2 and these definitions.

Lemma 6.6.3 *Let $\mathcal{X} = \{x_1, ..., x_n\}$ be a set of clocks, and let $\zeta \in \mathsf{Zone}(\mathcal{X})$ be a zone.*

1. *For any valuations $\mathbf{a},\mathbf{b} \in \mathbb{R}^n_{\geq 0}$, and zone $\zeta' \in \mathsf{Zone}(\mathcal{X})$, such that $\mathbf{a} \in \zeta$ and $\mathbf{b} \in \zeta'$, then $\mathsf{ZResets}(\zeta,\zeta') \subseteq \mathsf{SResets}(\mathbf{a},\mathbf{b})$.*
2. *For any distinct zones $\zeta',\zeta'' \in \mathsf{Zone}(\mathcal{X})$, then $\mathsf{ZResets}(\zeta,\zeta') \cap \mathsf{ZResets}(\zeta,\zeta'') = \emptyset$.*
3. *For any valuations $\mathbf{a},\mathbf{b} \in \mathbb{R}^n_{\geq 0}$, and zone $\zeta' \in \mathsf{Zone}(\mathcal{X})$, such that $\mathbf{a} \in \zeta$ and $\mathbf{b} \in \zeta'$, then*

$$\bigcup\{\mathsf{ZResets}(\zeta,\zeta') \mid \zeta' \in \mathsf{NewZones}(\zeta,\mathsf{SResets}(\mathbf{a},\mathbf{b}))\} = \mathsf{SResets}(\mathbf{a},\mathbf{b}) .$$

Part 2 of Lemma 6.6.3 implies that the elements of the set

$$\{\mathsf{ZResets}(\zeta,\zeta') \mid \zeta' \in \mathsf{NewZones}(\zeta,\mathsf{SResets}(\mathbf{a},\mathbf{b}))\}$$

are themselves *disjoint* sets. Therefore the union in part 3 of the lemma is a *disjoint* union. Hence, the set of sets $\{\mathsf{ZResets}(\zeta,\zeta') \mid \zeta' \in \mathsf{NewZones}(\zeta,\mathsf{SResets}(\mathbf{a},\mathbf{b}))\}$ is an $\emptyset$-inclusive partition of $\mathsf{SResets}(\mathbf{a},\mathbf{b})$.

We now proceed to the main lemma of this section.

Lemma 6.6.4 *Consider the probabilistic timed automaton T, the target set $Target \subseteq V$, and the forward reachability graph $\mathcal{S}_T^{Target}$. Let $(v,\mathbf{a}) \in S_T$ be a state of $\mathcal{S}_T$, and $(v,\zeta) \in S_T^{Target}$ be a symbolic state of $\mathcal{S}_T^{Target}$ such that $(v,\mathbf{a}) \in (v,\zeta)$. Consider the transition $(v,\mathbf{a}) \xrightarrow{\delta} \xrightarrow{\theta_p,\tilde{p}}$ of $\mathcal{S}_T$, the associated transition $(v,\zeta) \xrightarrow{\theta_p,\hat{p}}$ of $\mathcal{S}_T^{Target}$, and the state $(v',\mathbf{b}) \in S_T$ such that $\tilde{p}(v',\mathbf{b}) > 0$. Let $\mathcal{X}' \subseteq \mathcal{X}$ be the smallest set of clocks such that, for each $x \in \mathcal{X}'$, there exists some $X \subseteq \mathcal{X}'$ such that $p(v',X) > 0$ and $x \in X$, and consider the functions $\mathsf{SResets}$ and $\mathsf{ZResets}$ defined with respect to $\mathcal{X}'$. Then we have:*

$$\tilde{p}(v',\mathbf{b}) = \sum_{\zeta' \in \mathsf{NewZones}(\zeta \cap pre_v(p), \mathsf{SResets}(\mathbf{a}+\delta,\mathbf{b}))} \hat{p}(\mathsf{close}(\mathsf{time\_succ}(v',\zeta'),c))$$

Proof. From the definition of the distribution $\tilde{p}$ in Definition 4.3.2, and from the definition of $\mathsf{SResets}$:

$$\tilde{p}(v',\mathbf{b}) \;=\; \sum_{\substack{X \subseteq \mathcal{X}' \,\& \\ (\mathbf{a}+\delta)[X:=0]=\mathbf{b}}} p(v',X) \;=\; \sum_{X \in \mathsf{SResets}(\mathbf{a}+\delta,\mathbf{b})} p(v',X) \,. \tag{6.1}$$

Furthermore, by the definition of the distribution $\hat{p}$ in Definition 6.3.3, and from the definition of $\mathsf{ZResets}$, for each $\zeta' \in \mathsf{NewZones}(\zeta \cap pre_v(p), \mathsf{SResets}(\mathbf{a}+\delta, \mathbf{b}))$:

$$\begin{aligned}\hat{p}(\mathsf{close}(\mathsf{time\_succ}(v', \zeta'), c)) &= \sum_{\substack{X \subseteq \mathcal{X}' \,\&\\ (v',\zeta') = \mathsf{disc\_succ}((v,v',X,p)(v,\zeta))}} p(v', X) \\ &= \sum_{X \in \mathsf{ZResets}(\zeta \cap pre_v(p), \zeta')} p(v', X)\ .\end{aligned}$$

Summing over all zones in $\mathsf{NewZones}(\zeta \cap pre_v(p), \mathsf{SResets}(\mathbf{a}+\delta, \mathbf{b}))$ gives the following equation:

$$\begin{aligned}&\sum_{\zeta' \in \mathsf{NewZones}(\zeta \cap pre_v(p), \mathsf{SResets}(\mathbf{a}+\delta,\mathbf{b}))} \hat{p}(\mathsf{close}(\mathsf{time\_succ}(v', \zeta'), c)) \\ =\ &\sum_{\zeta' \in \mathsf{NewZones}(\zeta \cap pre_v(p), \mathsf{SResets}(\mathbf{a}+\delta,\mathbf{b}))} \left(\sum_{X \in \mathsf{ZResets}(\zeta \cap pre_v(p), \zeta')} p(v', X) \right) . \qquad (6.2)\end{aligned}$$

We now reconcile Equation 6.1 and Equation 6.2. The following equation relies on the crucial observation of Lemma 6.6.3 that the set of sets

$$\{\mathsf{ZResets}(\zeta \cap pre_v(p), \zeta') \mid \zeta' \in \mathsf{NewZones}(\zeta \cap pre_v(p), \mathsf{SResets}(\mathbf{a}+\delta, \mathbf{b}))\}$$

is an $\emptyset$-inclusive partition of $\mathsf{SResets}(\mathbf{a}+\delta, \mathbf{b})$, and Equation 3.1:

$$\begin{aligned}&\sum_{X \in \mathsf{SResets}(\mathbf{a}+\delta,\mathbf{b})} p(v', X) \\ =\ &\sum_{\zeta' \in \mathsf{NewZones}(\zeta \cap pre_v(p), \mathsf{SResets}(\mathbf{a}+\delta,\mathbf{b}))} \left(\sum_{X \in \mathsf{ZResets}(\zeta \cap pre_v(p), \zeta')} p(v', X) \right) .\end{aligned}$$

Therefore, from Equation 6.1 and Equation 6.2, after performing appropriate substitutions:

$$\tilde{p}(v', \mathbf{b}) = \sum_{\zeta' \in \mathsf{NewZones}(\zeta \cap pre_v(p), \mathsf{SResets}(\mathbf{a}+\delta,\mathbf{b}))} \hat{p}(\mathsf{close}(\mathsf{time\_succ}(v', \zeta'), c))\ . \qquad \square$$

We conclude this section with the following observation. For a distribution $\tilde{p}$ of the concurrent probabilistic system $\mathcal{S}_T$, there will be a finite number $l \in \mathbb{N}$ of states in the support of $\tilde{p}$; then, for a related distribution $\hat{p}$ of the forward reachability graph $\mathcal{S}_T^{Target}$, which is derived from the same distribution of T, there will be *at least* l symbolic states within the support of $\hat{p}$. Intuitively, the transitions of the forward reachability graph can be though of as "probabilistically branching more than" the associated transitions of the concurrent probabilistic system.

6.6.2 Properties of adversaries

The properties of Section 6.6.1 can be extended to reason about the correspondence between the paths and adversaries of both the concurrent probabilistic system $\mathcal{S}_T$ and the forward reachability graph $\mathcal{S}_T^{Target}$.

Adversary construction process

The lemmas of Section 6.6.1 suggest the following result: for any adversary A of $\mathcal{S}_T$, an adversary B of $\mathcal{S}_T^{Target}$ can be constructed. This construction process will proceed by starting from the initial states of $\mathcal{S}_T$ and $\mathcal{S}_T^{Target}$, progressing along the length of paths of A and successively adding transitions to the partially constructed paths of B.

As the construction process, and the subsequent proof of correctness, both proceed by induction on the length of paths in $Path^A_{fin}(\bar{v}, \mathbf{0})$, we slightly redefine our notion of the length $|\omega|$ of a path $\omega \in Path^A_{fin}(\bar{v}, \mathbf{0})$ of the concurrent probabilistic system $\mathcal{S}_T$ to accommodate the fact that we now regard a transition as taking the form $(v, \mathbf{a}) \xrightarrow{\delta} \xrightarrow{\theta_p, \tilde{p}} (v, \mathbf{b})$. That is, if a finite path $\omega \in Path^A_{fin}(\bar{v}, \mathbf{0})$ is of the form

$$\omega = (v_0, \mathbf{a}_0) \xrightarrow{\delta_0} \xrightarrow{\theta_{p_0}, \tilde{p}_0} (v_1, \mathbf{a}_1) \xrightarrow{\delta_1} \xrightarrow{\theta_{p_1}, \tilde{p}_1} \ldots \xrightarrow{\delta_{n-1}} \xrightarrow{\theta_{p_{n-1}}, \tilde{p}_{n-1}} (v_n, \mathbf{a}_n) \ ,$$

then we now let $|\omega| = n$ (instead of $|\omega| = 2n$).

We proceed by induction on the length of paths of $\omega \in Path^A_{fin}(\bar{v}, \mathbf{0})$. Our aim is to construct the set of paths $\Pi_\omega \subseteq Path_{fin}(\langle \bar{s} \rangle)$, defining the choices of the adversary B as we proceed along their length so that these choices mimic the transitions along ω. Note that paths in the set Π_ω will be the same length as ω.

The induction hypothesis for ω and Π_ω takes the following form. Either:

1. there exists $i \leq |\omega|$ such that $\mathsf{discrete}(\omega(i)) \in Target$, or
2. for each $i \leq |\omega|$, then the ith state along ω, denoted by $\omega(i)$, will be contained within the ith symbolic state $\pi(i)$ of all paths $\pi \in \Pi_\omega$.

Our base case concerns paths of length 0; that is, paths comprising only the initial state $(\bar{v}, \mathbf{0})$ and $\langle \bar{s} \rangle$. As $\langle \bar{s} \rangle = \mathsf{close}(\mathsf{time\_succ}(\bar{v}, \mathbf{0}), c)$, it follows that $(\bar{v}, \mathbf{0}) \in \langle \bar{s} \rangle$, trivially satisfying condition 2 of the induction hypothesis.

Next, consider any path $\omega \in Path^A_{fin}(\bar{v}, \mathbf{0})$ of length $i + 1$. Then ω is of the form $\omega' \xrightarrow{\delta} \xrightarrow{\theta_p, \tilde{p}} (v, \mathbf{a})$ for some path ω' of length i and $(v, \mathbf{a}) \in S_T$. By induction we have constructed the set of paths $\pi \in \Pi_{\omega'}$, and we show how to extend this set to form Π_ω. We consider the following cases:

Case: ***Target*** **has already been reached.** That is, there exists a $j \leq i$ such that $\mathsf{discrete}(\omega'(j)) \in Target$. Let $k = \min\{j \leq i \mid \mathsf{discrete}(\omega'(j)) \in Target\}$ be the index of the first state along ω' such that the location is in $Target$. Then, from the induction hypothesis (recall that this says that a state along a path ω' is contained within a symbolic state at the same point along the path $\pi' \in \Pi_{\omega'}$), we conclude that $\mathsf{discrete}(\pi'(k)) = \mathsf{discrete}(\omega'(k))$. From the definition of the forward reachability graph (Definition 6.3.3), once a state of $\mathcal{S}_T^{\mathit{Target}}$ has reached a location in *Target*, then it loops in its current symbolic state. Therefore, regardless of the transition made by A after ω', we let $B(\pi') = (\bot, \mathcal{D}(\mathit{last}(\pi')))$ for all $\pi' \in \Pi_{\omega'}$, and define Π_ω as the set of paths $\{\pi' \xrightarrow{\bot, \mathcal{D}(\mathit{last}(\pi'))} \mathit{last}(\pi') \mid \pi' \in \Pi_{\omega'}\}$.

Note that the fact that *Target* has already been reached means that part 1 of the induction hypothesis is preserved.

Case: ***Target* has not already been reached.** Consider any path $\pi' \in \Pi_{\omega'}$ and say $\pi'(i) = (v', \zeta_{\pi'})$ and $\omega'(i) = (v', \mathbf{b})$. From part 2 of the induction hypothesis, we have $(v', \mathbf{b}) \in (v', \zeta_{\pi'})$. Then from this fact and Lemma 6.6.1, there exists $(v', \zeta_{\pi'}) \xrightarrow{\theta_p, \hat{p}_{\pi'}}$. We simply let $B(\pi') = (\theta_p, \hat{p}_{\pi'})$.

Now we must consider the preservation of the induction hypothesis by the transition; that is, we extend π' by a set of transitions for which we are certain that the target symbolic state contains $(v, \mathbf{a})$.

Observe that the subscript for the event θ_p is a distribution $p \in \mathit{prob}(v)$. By Lemma 6.6.1, there exists a symbolic state (v, ζ) in the support of $\hat{p}_{\pi'}$ such that $(v, \mathbf{a}) \in (v, \zeta)$. Furthermore, for all $\zeta \in \mathsf{NewZones}(\zeta' \cap \mathit{pre}_{v'}(p), \mathsf{SResets}(\mathbf{b}+\delta, \mathbf{a}))$, it must be the case that $\mathbf{a} \in \zeta$. Therefore $(v, \mathbf{a}) \in \mathsf{close}(\mathsf{time\_succ}(v, \zeta), c)$. Then we extend the path π' to the set of paths of the form

$$\pi' \xrightarrow{\hat{p}_{\pi'}} \mathsf{close}(\mathsf{time\_succ}(v, \zeta), c)$$

such that

$$\zeta \in \mathsf{NewZones}(\zeta' \cap \mathit{pre}_{v'}(p), \mathsf{SResets}(\mathbf{b} + \delta, \mathbf{a})\}) \, .$$

Note that for any such ζ, by Definition 6.3.2, we have $\mathsf{close}(\mathsf{time\_succ}(v, \zeta), c) = \mathsf{post}[(v', v, X, p), c](v', \zeta')$ for some $(v', v, X, p) \in \mathsf{out}(v')$.

Then, to construct the set Π_ω, we take the union of these extensions over all $\pi' \in \Pi_{\omega'}$.

Furthermore, repeating the construction procedure for all $\omega \in \mathit{Path}_{\mathit{fin}}^A(\bar{v}, \mathbf{0})$ results in an adversary B on $\mathcal{S}_T^{\mathit{Target}}$. We say that B is *constructed* from A.

Reachability probability of adversaries

Next, we present the fundamental property of B: that the probability with which it reaches the target set *Target* of locations is equal to the probability of A reaching the set *Target*.

Lemma 6.6.5 *For the probabilistic timed automaton T, the target set of locations $Target \subseteq V$, and the forward reachability graph $\mathcal{S}_T^{Target}$, let A be an adversary of $\mathcal{S}_T$, and B be the adversary of $\mathcal{S}_T^{Target}$ constructed from A. If $\omega \in Path_{fin}^A(\bar{v}, \mathbf{0})$ and $\mathsf{discrete}(\omega(i)) \notin Target$ for all $i < |\omega|$, then*

$$Prob_{fin}(\omega) = Prob'_{fin}(\Pi_\omega) .$$

Proof. The proof proceeds by induction on the length of ω. If $|\omega| = 0$, then by definition of $Prob_{fin}$ and $Prob'_{fin}$, and by construction of Π_ω, we have $Prob_{fin}(\omega) = Prob'_{fin}(\Pi_\omega) = 1$ as required.

Now suppose that the lemma holds for all paths of length at most $k \in \mathbb{N}$, and consider any path $\omega \in Path_{fin}^A(\bar{v}, \mathbf{0})$, for which $|\omega| = k+1$ such that $\mathsf{discrete}(\omega(i)) \notin Target$ for all $i < k+1$. Let $\omega' = \omega^{(k)}$. If the adversary A is such that $A(\omega') = (\delta, \mathcal{D}(last(\omega') + \delta))$, and $A(\omega' \xrightarrow{\delta} last(\omega') + \delta) = (\theta_p, \tilde{p})$, then $\omega = \omega' \xrightarrow{\delta} \xrightarrow{\theta_p, \tilde{p}} (v, \mathbf{a})$ for some $(v, \mathbf{a}) \in S_T$. Note that $|\omega'| = k$ and, from the fact that $\mathsf{discrete}(\omega(i)) \notin Target$ for all $i < |\omega|$, we also have that $\mathsf{discrete}(\omega'(i)) \notin Target$ for all $i < |\omega'|$. Hence, by induction, we have:

$$Prob_{fin}(\omega') = Prob'_{fin}(\Pi_{\omega'}) . \tag{6.3}$$

For any path $\pi' \in \Pi_{\omega'}$, let $last(\pi') = (v', \zeta_{\pi'})$ and let $(v', \zeta_{\pi'}) \xrightarrow{\theta_p, \hat{p}}$ be a transition, which exists by Lemma 6.6.1.

Now suppose that $last(\omega') = (v', \mathbf{b})$. Then by the definition of $Prob'_{fin}$ and the construction of B, we have:

$$Prob'_{fin}(\Pi_\omega) = \\ \sum_{\pi' \in \Pi_{\omega'}} Prob'_{fin}(\pi') \cdot \left(\sum_{\zeta \in \mathsf{NewZones}(\zeta_{\pi'} \cap pre_{v'}(p), \mathsf{SResets}(\mathbf{b}+\delta, \mathbf{a}))} \hat{p}_{\pi'}(\mathsf{close}(\mathsf{time\_succ}(v, \zeta), c)) \right) .$$

Furthermore, by Lemma 6.6.4 and the construction of B, for any $\pi \in \Pi_\omega$:

$$\tilde{p}(v, \mathbf{a}) = \sum_{\zeta \in \mathsf{NewZones}(\zeta_{\pi'} \cap pre_{v'}(p), \mathsf{SResets}(\mathbf{b}+\delta, \mathbf{a}))} \hat{p}_{\pi'}(\mathsf{close}(\mathsf{time\_succ}(v, \zeta), c)) .$$

Combining these two facts, we have:

$$\begin{array}{rcll} Prob'_{fin}(\Pi_\omega) & = & \sum_{\pi' \in \Pi_{\omega'}} Prob'_{fin}(\pi') \cdot \tilde{p}(v, \mathbf{a}) & \\ & = & \left(\sum_{\pi' \in \Pi_{\omega'}} Prob'_{fin}(\pi') \right) \cdot \tilde{p}(v, \mathbf{a}) & \\ & = & Prob'_{fin}(\Pi_{\omega'}) \cdot \tilde{p}(v, \mathbf{a}) & \text{(by definition of } Prob'_{fin}\text{)} \\ & = & Prob_{fin}(\omega') \cdot \tilde{p}(v, \mathbf{a}) & \text{(by Equation 6.3)} \\ & = & Prob_{fin}(\omega' \xrightarrow{\delta} \xrightarrow{\theta_p, \tilde{p}} (v, \mathbf{a})) & \text{(by definition of } Prob_{fin}\text{)} \\ & = & Prob_{fin}(\omega) & \text{(by construction)} \end{array}$$

as required. □

Proposition 6.6.6 *For the probabilistic timed automaton T, the target set of locations $\mathit{Target} \subseteq V$, and the forward reachability graph $\mathcal{S}_T^{\mathit{Target}}$, let A be an adversary of $\mathcal{S}_T$, and B be the adversary of $\mathcal{S}_T^{\mathit{Target}}$ constructed from A. Then:*

$$\begin{aligned} & \mathit{Prob}^A(\{\omega \in \mathit{Path}^A_{\mathit{ful}}(\bar{v}, \mathbf{0}) \mid \exists i \in \mathbb{N}.\mathsf{discrete}(\omega(i)) \in \mathit{Target}\}) \\ = \; & \mathit{Prob}^B(\{\pi \in \mathit{Path}^B_{\mathit{ful}}(\langle \bar{s} \rangle) \mid \exists i \in \mathbb{N}.\mathsf{discrete}(\pi(i)) \in \mathit{Target}\}) \, . \end{aligned}$$

Proof. The proof follows from Lemma 6.6.5 and the fact that the probability measures Prob^A and Prob^B are induced from the functions $\mathit{Prob}^A_{\mathit{fin}}$ and $\mathit{Prob}^B_{\mathit{fin}}$ respectively. □

Correspondence between divergent adversaries

Given our definition of *Target*-divergent adversaries of the concurrent probabilistic system $\mathcal{S}_T$ (Definition 6.5.2), and the definition of *Target*-divergent adversaries of the forward reachability graph $\mathcal{S}_T^{\mathit{Target}}$ (Definition 6.5.6), with respect to the probabilistic timed automaton T and the target set of locations $\mathit{Target} \subseteq V$, we now show that the adversary construction procedure presented in Section 6.6.2 preserves desired properties of divergence.

Proposition 6.6.7 *Let T be a probabilistic timed automaton, $\mathit{Target} \subseteq V$ be a target set of locations, and let $\mathcal{S}_T^{\mathit{Target}}$ be the associated forward reachability graph. For any Target-divergent adversary A of $\mathcal{S}_T$, the adversary B of $\mathcal{S}_T^{\mathit{Target}}$ that is constructed from A is Target-divergent.*

Proof. We proceed by showing that, if the path $\omega \in \mathit{Path}^A_{\mathit{fin}}(\bar{v}, \mathbf{0})$ of A is *Target*-divergent, then all paths $\pi \in \Pi_\omega$ are also *Target*-divergent (where Π_ω is the set of paths introduced in the construction procedure for B). Firstly, we consider the case in which there exists $i \in \mathbb{N}$ such that $\mathsf{discrete}(\omega(i)) \in \mathit{Target}$; that is, the path ω reaches a target location. Then, by the construction procedure for B, for all paths $\pi \in \Pi_\omega$, we have $\mathsf{discrete}(\pi(i)) = \mathsf{discrete}(\omega(i))$. Therefore, $\mathsf{discrete}(\pi(i)) \in \mathit{Target}$, and π is *Target*-divergent.

Secondly, we consider the case in which the path ω does not reach the target set *Target*. Consider the forward reachability graph path $\pi \in \Pi_\omega$. Firstly, as ω is divergent, then π must not be zero, as zero paths feature a clock which does not advance from some point onwards, therefore blocking time.

Now, as π is not zero, we must establish that it is possibly progressive for π to be divergent. Say that there exists a clock $x \in \mathcal{X}$ such that x is *not* reset from the ith transition in π onwards, for some $i \in \mathbb{N}$; that is, for the clock x, the path π does not satisfy part 1 of the definition of possibly progressive forward reachability graph paths (Definition 6.5.5). Then, as the path ω is divergent, and therefore corresponds to time passing above any bound, it must be the case that the value of x is not constrained by either an invariant condition or an enabling condition which is associated with a distribution of T taken at some point along ω from the ith transition onwards. Therefore, condition 2 of the definition of possibly progressive forward reachability graph paths must be satisfied by π. Hence, the path π is then *Target*-divergent. □

6.6.3 Correctness of probabilistic real-time reachability: conclusions

From the results of Section 6.6.2, we conclude that, for the target set of locations *Target*, and for any *Target*-divergent adversary *Target* of the concurrent probabilistic system $\mathcal{S}_T$, we can construct an *Target*-divergent adversary of the forward reachability graph $\mathcal{S}_T^{\mathit{Target}}$ such that the probability of reaching a location in *Target* are the same for both A and B.

Therefore, for the adversary of $\mathcal{S}_T$ with the maximal probability of reaching a location in *Target*, there exists an adversary of $\mathcal{S}_T^{\mathit{Target}}$ with the same probability of reaching *Target*. Hence, the maximal reachability probability computed on $\mathcal{S}_T^{\mathit{Target}}$ must be an upper bound on the maximal probability of reaching a location in *Target* for $\mathcal{S}_T$.

Chapter 7

Probabilistic hybrid automata

7.1 Introduction

This chapter introduces a formalism for the description of hybrid systems for which relative likelihoods are associated with certain behaviours. We henceforth refer to such systems as *probabilistic hybrid systems*. As an example of such a system, consider the following, which is adapted from the non-probabilistic hybrid automata literature [HHM99, HM00b]. A factory contains a production line which consists of a conveyor belt divided into two sections, the length of each being 3 metres. An item arrives at the line at most once every 4 minutes; upon arrival, it is placed on the first section of the conveyor belt, along which it proceeds at between 2 and 3 metres per minute. As an item passes between the sections of the conveyor belt, there is a 1% chance of a the occurrence of a fault, which necessitates production being halted for two minutes. After this time, or if a fault did not occur originally, the item progresses at between 1 and 4 metres per minute along the second segment of the conveyor belt, until it reaches the end of this section. However, if a new job arrives before the previous one was completed, the production line undergoes a complete shutdown. A pictorial representation of the system is shown in Figure 7.1.

Although the system has aspects concerning timing, such as the arrival rate of jobs, the direct modelling of the position of an item along either segment of the conveyor belt is more problematic. We discuss in Chapter 8 a number of methods of representing such behaviour by (probabilistic) timed automata in restricted settings; however, in general, (probabilistic) timed automata are unsuitable for modelling such continuous aspects of a system. Therefore, we must search for a more convenient framework for the modelling of such systems.

One of the dominant formalisms for the specification and verification of (non-probabilistic) hybrid systems is that of *hybrid automata* [ACHH93, NOSY93]. As explained in Section 2.4, this framework generalises that of timed automata in allowing more general conditions on system variables that allows discrete transitions to be

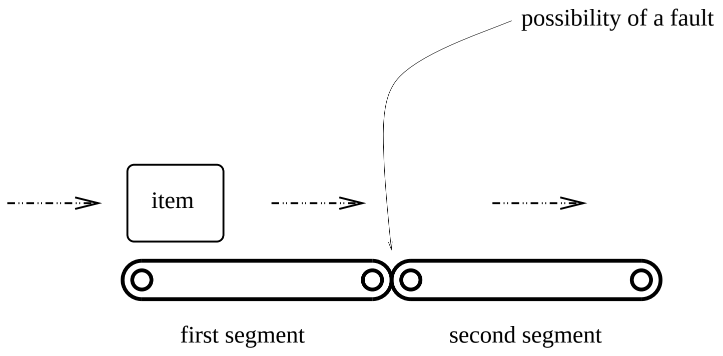


Figure 7.1: A faulty production line.

enabled or disabled, as well as permitting such variables to increase or decrease according to more complex differential equations or inequalities. As already stated, a hybrid system similar to the one described above has been modelled and verified using hybrid automata [HHM99, HM00b]. As an example of how differential inequalities can be used to aid the direct modelling of this hybrid system, consider the following: let x be a variable which denotes the distance which the item has progressed along the conveyor belt. Then the rate of an item's progress along the first segment of the belt can be written as the interval [2,3], with the intuition that the first derivative of x with respect to time, written $\dot{x}$ as shorthand for $\frac{dx}{dt}$ where t represents time, lies within this interval.

Given such a formalism, which both expands on timed automata and is amenable to a number of model checking techniques, we are now interested in its *probabilistic* extension in the style of probabilistic timed automata. That is, we extend the discrete transition relation of hybrid automata with probabilistic information in the style of concurrent probabilistic systems or Markov decision processes. Therefore, the aim of this chapter is to define the syntax and semantics of probabilistic hybrid automata, and to present a more complex example of the way in which such models may be used to describe probabilistic hybrid systems. Parts of this chapter have already appeared in [Spr99, Spr00].

7.2 Syntax of probabilistic rectangular automata

7.2.1 Definition of probabilistic rectangular automata

We now define our model for probabilistic hybrid systems, which we call *probabilistic hybrid automata*. Emphasis is placed on a number of restricted subclasses of probabilistic hybrid automata, using as our precedent a number of subclasses of non-probabilistic hybrid automata that have been described previously in the literature. Recall from

Section 2.4 that such a classification is made according to the expressiveness of both the discrete and continuous behaviour of the model. Our main focus will concern generalisations of timed automata, from *stopwatch automata* and *multisingular automata* [ACH$^+$95] to *rectangular automata* [PV94, HKPV98]. We now present a probabilistic extension of rectangular automata, the most general of these classes. Note that a number of concepts and notation in the following definition are borrowed from the definition of probabilistic timed automata in Chapter 4, and that a wider class of probabilistic hybrid automata are defined in [Spr00].

Definition 7.2.1 (Probabilistic rectangular automaton) *A* probabilistic rectangular automaton $R = (\mathcal{X}, V, \mathcal{L}, init, inv, flow, prob, \langle pre_v \rangle_{v \in V})$ *comprises the following components:*

Variables. $\mathcal{X}$ *is a finite set of real-valued* variables. *(If* $\mathcal{X}$ *contains n elements, we say that R is an n-dimensional (or nD) probabilistic rectangular automaton.)*

Locations. V *is a finite set of* locations.

Labelling function. *The function* $\mathcal{L} : V \to 2^{\mathrm{AP}}$ *assigns a finite set of atomic propositions to each location.*

Initial set. *The function* $init : V \to \mathsf{Rect}(\mathcal{X})$ *maps every location to an* initial set *in* $\mathsf{Rect}(\mathcal{X})$.

Invariant set. *The function* $inv : V \to \mathsf{Rect}(\mathcal{X})$ *maps every location to an* invariant set *in* $\mathsf{Rect}(\mathcal{X})$.

Flow inclusion. *The function* $flow : V \to \mathsf{Rect}(\mathcal{X})$ *maps every location to a* flow inclusion *in* $\mathsf{Rect}(\mathcal{X})$.

Probability distributions. *The function* $prob : V \to 2_{fn}^{(\mu(V \times \mathsf{Rect}(\mathcal{X}) \times 2^{\mathcal{X}}))}$ *maps every location to a finite, non-empty set of discrete probability distributions over the set of locations, the set of rectangles over* $\mathcal{X}$*, and the powerset of* $\mathcal{X}$*. Therefore, each location v will have a set of associated probability distributions, denoted by* $prob(v) = \{p_v^1, ..., p_v^m\}$ *for some finite* $m \geq 1$.

Pre-condition sets. *For each* $v \in V$*, the function* $pre_v : prob(v) \to \mathsf{Rect}(\mathcal{X})$ *maps every probability distribution associated with a location to a* pre-condition set *in* $\mathsf{Rect}(\mathcal{X})$.

For simplicity, and without loss of generality, we assume that the initial point is unique; that is, for a control mode $v \in V$, $init(v) \in \mathsf{Rect}(\mathcal{X})$ is a singleton, and for all other locations $v' \in V \setminus \{v\}$, we have $init(v') = \emptyset$.

For notational convenience, we write flow conditions in the following manner. First recall that, for any rectangle $\gamma \in \mathsf{Rect}(\mathcal{X})$, we use γ_i to denote the projection of γ onto the axis of the variable $x_i \in \mathcal{X}$. For the variable $x_i \in \mathcal{X}$, consider the flow condition $\mathit{flow}(v)_i = I$, where I is some open, closed, partially open, or bounded or unbounded interval of the real line. Then we write $\dot{x} \in I$ to represent this flow condition.

Another subclass of hybrid automata which has been the subject of much analysis, particularly in the context of model checking, is that of *linear hybrid automata*. However, the verification algorithm presented in [AHH96], and implemented in the tool HyTech [HHW97], is based on explorative traversal of the state space, rather than reduction to timed automata. We expect that a method for model checking a probabilistic extension of linear hybrid automata against temporal logic properties would take the form of an extension of the explorative algorithm of [KNS00] (in the same way that [AHH96] is an extension of the algorithm of [HNSY94] for timed automata to linear hybrid automata).

7.2.2 Behaviour of probabilistic rectangular automata

The behaviour of a probabilistic rectangular automaton takes the form of transitions between its locations which are both nondeterministic and probabilistic in nature. These transitions are dependent on the values of the system's variables, in exactly the same manner as for probabilistic timed automata. However, the fact that the variables of probabilistic rectangular automata reflect a wider class of continuous phenomena than simply the passage of time allows us to express behaviour such as, "when the item has progressed 3 metres along the production line, then a fault occurs with probability 0.01". As with probabilistic timed automata, discrete transitions are determined by comparing the current values of the variables with invariant conditions and pre-conditions; that is, if control resides in a location v, a transition can be made according to the distribution $p \in \mathit{prob}(v)$ if the current values of the variables satisfy $\mathit{pre}_v(p)$, and a probabilistic transition must be made if the advancement of time would result in the violation of the invariant condition $\mathit{inv}(v)$. Another similarity with our previous model is that pre-conditions are associated with distributions, as opposed to individual edges (this assumption is made for the same reasons as in the case of probabilistic timed automata). Note that distributions of a probabilistic hybrid automaton are over tuples of the form (v, post, X); in the standard manner for probabilistic timed automata, v is the location to which control will now pass, and the variables in X are reset according to the values given by post. However, we now permit post to be a rectangle, with the intuition that any variable $x_i \in X$ can nondeterministically select a value in the (closed or open, bounded or unbounded) interval post_i. As before, the values of all variables in $\mathcal{X}$ but not in X remain the same.

Naturally, the continuous transitions made by probabilistic rectangular automata

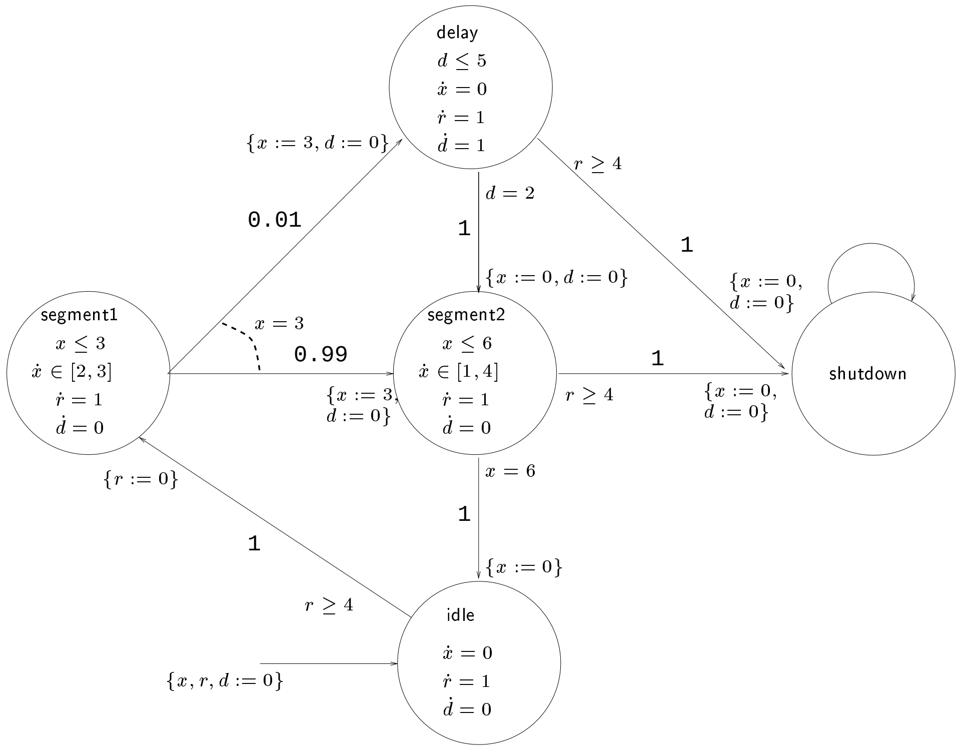


Figure 7.2: The probabilistic rectangular automaton R_{pl}.

are more complex than those of probabilistic timed automata. Instead of letting all variables increase at the same rate as real-time, so that a time transition of duration $\delta \in \mathbb{R}_{\geq 0}$ results in the addition of δ to the value of each variable, the rate of change of variables in a location v must fall within the bounds given by the rectangular flow inclusion $\mathit{flow}(v)$. Therefore, for a variable $x_i \in \mathcal{X}$, if $\mathit{flow}(v)_i$ is not a singleton, then the rate of change of x_i is dependent on a *nondeterministic* choice.

7.2.3 A probabilistic rectangular automaton for the production line

The probabilistic hybrid system of the production line presented in Section 7.1 can be modelled as the probabilistic rectangular automaton R_{pl}, as shown in Figure 7.2. The locations are idle, which represents the situation in which no job is being processed, segment1 and segment2, in which an item is being successfully processed along the first and second sections of the conveyor belt respectively, delay, in which the production line is halted because of a fault, and shutdown. The clock r models the elapsed time since the arrival of the last item, the continuous variable x represents the distance in metres which the current item has progressed along the production line, and the stopwatch d is used in the presence of a fault to model the time elapsed since production has

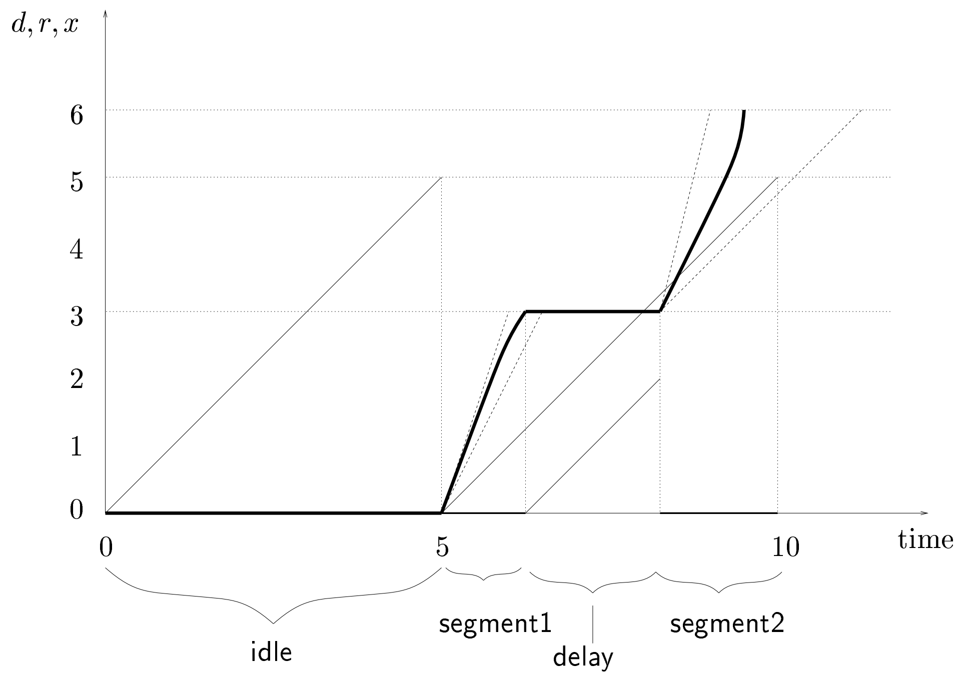


Figure 7.3: An example run of the probabilistic rectangular automaton R_{pl}.

been halted. In the usual manner for hybrid automata, invariant conditions (such as $x \leq 3$) and flow conditions (such as $\dot{x} \in [2,3]$) are written in the body of locations, and edges are labelled with their pre-conditions (for example, $r \geq 4$) and post-conditions (for example, $\{x := 0\}$). The edges joined by the dotted arc are the only two which are probabilistic; the edge whose target is delay represents the occurrence of a fault, and as such is labelled with probability 0.01, whereas the edge with segment2 as its target represents successful progression to the next stage of the production process, and is labelled with the probability 0.99. All other probability distributions assign probability 1 to a single edge. Omitted for simplicity are invariants of the form `true`, and the continuous dynamics of the location shutdown.

An example computation of this probabilistic rectangular automaton is shown in Figure 7.3. The values of the variables x, r and d label the y-axis of the figure, and time is represented on the x-axis. The latter axis is also labelled with the locations in which control of the probabilistic rectangular automaton resides as time passes. The execution commences in the location idle with the values of all variables being 0. As time passes, the values of the variables x and d remain constant, while the value of the clock r increases until it reaches 5. At this point, an item arrives, and is placed on the first conveyor belt. This is denoted by a control switch to the location segment1, which involves the clock r being reset to 0. The value of x now increases according to the flow constraint denoted by $\dot{x} \in [2,3]$; the bounds on the rate of increase of this variable are denoted by the dashed lines in this part of Figure 7.3, with the actual trajectory being the solid line between them. The continuous dynamics of the variables r and d remain as in the location idle.

When the value of x reaches 3 (corresponding to the item reaching the end of the first conveyor belt segment), the invariant of the location $\mathsf{segment1}$, together with the pre-condition of the only distribution associated with this location, force a probabilistic choice over the edges to the locations delay and $\mathsf{segment2}$. Say the former edge is chosen. In the location delay, the value of x now remains constant (corresponding to production being halted), while the variables r and now d increase at the same rate as real-time. When the value of d reaches 2, corresponding to 2 minutes elapsing since production was halted, control then moves to the location $\mathsf{segment2}$; that is, production is recommenced. Therefore, the rate of change of x is now given by $\dot{x} \in [1, 4]$; again these bounds on the first derivative of this variable with respect to time is represented in Figure 7.3 by dashed lines, whereas the actual trajectory is the solid line contained between these bounds. As the value of x then reaches 6 before another item arrives at the production line, then control returns to the location idle. This cycle of behaviour then continues in much the same fashion, although it may be possible that, if the value of the clock r exceeds 4 when control resides in one of the locations delay or $\mathsf{segment2}$, then another item may arrive at the production line, therefore necessitating a shutdown.

7.2.4 Well-formedness assumptions

We expand the notion of tightened pre-conditions of probabilistic timed automata to the hybrid context. Recall from Section 4.2.4 that this requirement alters the pre-conditions of a given probabilistic timed automaton so that, given the choice of a particular enabled distribution, for any probabilistic selection of an edge of this distribution, the subsequent target state is admissible; that is, it is contained in the appropriate invariant set. In the case of probabilistic rectangular automata, the possibility of non-singular post conditions of tuples in the support of a particular distribution makes matters more involved.

First we introduce a notion of *tightened post conditions*. This involves operations on the post conditions of every tuple $(v, post, X)$ in the support of distribution of a probabilistic rectangular automaton. More precisely, we make sure that any valuation chosen according to X and $post$ is in the target invariant set $inv(v)$.

Definition 7.2.2 *For a probabilistic rectangular automata R, any location $v \in V$, distribution $p \in prob(v)$, and tuple $(v', post, X) \in \mathsf{support}(p)$, we let the* tightened post condition *$post' \in \mathsf{Rect}(\mathcal{X})$ of $(v', post, X)$ be a rectangle satisfying $post'_i = post_i \cap inv(v')_i$ for each variable $x_i \in X$.*

It turns out that, for the variables $x_i \notin X$, the value of $post'_i$ may be arbitrary. We henceforth assume that all of the post conditions of any rectangular automaton R are tightened.

Next, for the rectangles $post, \gamma \in \mathsf{Rect}(\mathcal{X})$ and the set of variables $X \subseteq \mathcal{X}$, recall from Section 3.4 that $[X := post]\gamma$ denotes the rectangle containing the valuations $\mathbf{a} \in \mathbb{R}^n$ for which the rectangle $\mathbf{a}[X := post]$ is contained within γ. Intuitively, from a valuation in $[X := post]\gamma$, it is possible to reset the variables in X to result in a valuation in γ, given the nondeterministic choice from $post$ of values to which variables are set. Then the intersection of rectangles of the form $[X := post]inv(v')$ for all tuples $(v', post, X) \in \mathsf{support}(p)$, for a particular distribution p, denotes the set of valuations for which it is possible to reach an admissible state, no matter which tuple is probabilistically chosen. This set is then intersected with the pre-condition of the distribution to result in a new, tightened pre-condition. Note that the tightened pre-condition will also be a rectangle, as all of the valuation sets described above are also rectangles, and the intersection of a finite number of rectangles is also a rectangle.

Definition 7.2.3 *For a probabilistic rectangular automata R, any location $v \in V$ and distribution $p \in prob(v)$ with the pre-condition $pre_v(p)$, the* tightened pre-condition *$pre'_v(p)$ of p is the zone:*

$$pre'_v(p) = pre_v(p) \cap inv(v) \cap \bigcap_{(v',post,X) \in \mathsf{support}(p)} [X := post]inv(v') \ .$$

Henceforth, we assume that the pre-conditions of all of the locations of probabilistic rectangular automata are tightened.

7.2.5 Subclasses of probabilistic rectangular automata

We now introduce some terminology which we use to classify probabilistic rectangular automata, all of which is adapted from the non-probabilistic precedent of [HKPV98]. Certain results presented in the sequel are dependent on the imposition of two restrictions on the discrete jumps of probabilistic rectangular automata, which are called *initialisation* and *deterministic jumps.* The probabilistic rectangular automaton R is *initialised* if, for every pair of locations $v, v' \in V$, and every variable $x_i \in \mathcal{X}$ for which $flow(v)_i \neq flow(v')_i$, then if there exists a distribution $p \in prob(v)$ and a tuple $(v', post, X) \in \mathsf{support}(p)$, then it must be the case that $x_i \in X$. Intuitively, if the execution of a discrete transition results in a variable changing the condition on its continuous evolution, then the value of that variable must be reinitialised. The probabilistic rectangular automaton R has *deterministic jumps* if for the locations $v, v' \in V$, distribution $p \in prob(v)$, and tuple $(v', post, X) \in \mathsf{support}(p)$, then, for every $x_i \in X$, the rectangle $post_i$ is a singleton. Intuitively, this requirement states that, for every discrete transition, each variable either remains unchanged or is deterministically reset to a new value.

A *probabilistic multisingular automaton* M is an initialised probabilistic rectangular automaton with deterministic jumps such that, for each $v \in V$ and for each $x_i \in \mathcal{X}$, we have $flow(v)_i = [k, k]$ for some $k \in \mathbb{N}$. A *probabilistic stopwatch automaton* W is a probabilistic multisingular automaton such that, for each location $v \in V$ and for each variable $x_i \in \mathcal{X}$, either $flow(v)_i = [1, 1]$ or $flow(v)_i = [0, 0]$. A probabilistic stopwatch automaton such that $flow(v)_i = 1$, for each location $v \in V$ and for each variable $x_i \in \mathcal{X}$ is a probabilistic timed automaton T. However, there are some probabilistic timed automata (with or without integer resets) that cannot be expressed as probabilistic stopwatch automata. This is simply because we permit *zones* from the set $\mathsf{Zone}(\mathcal{X})$ in the invariant conditions and pre-conditions of probabilistic timed automata, whereas such conditions in probabilistic stopwatch automata only feature rectangles from $\mathsf{Rect}(\mathcal{X})$. Taking the syntactic view of zones and rectangles as conjunctions of constraints, then observe that rectangles forbid constraints of the form $x - y \sim k$, where $x, y \in \mathcal{X}$ are variables, $k \in \mathbb{N}$ is a natural number constant, and $\sim \in \{<, \leq, \geq, >\}$ is a comparison operator. Also note that we require non-negativity of the probabilistic stopwatch automaton because zones are not described in terms constraints referring to negative constants. It has been shown in the non-probabilistic setting [HKPV98] that, in general (with a few notable exceptions, such as those featured in [KPSY99, ACH97]), the reachability problem is undecidable for a rectangular automaton with clocks and a single stopwatch, and which features invariants and pre-conditions in the form of zones.

7.3 Semantics of probabilistic rectangular automata

The semantics of a given probabilistic rectangular automaton R can be represented in terms of concurrent probabilistic systems in the following way. A concurrent probabilistic system for which the duration of continuous transitions is observable is defined first, allowing us to reason about the accumulated duration of time of certain behaviours. A concurrent probabilistic system for which the duration of such transitions is abstracted is then presented. This follows the precedent of the definition of standard and time-abstract concurrent probabilistic systems or probabilistic timed automata in Chapter 4.

The following notation is used to reason about the target states of the probabilistic transitions of R. Recall the definition of the rectangle $\mathbf{a}[X := \gamma]$, where $\mathbf{a} \in \mathbb{R}^n$ is a valuation, $\gamma \in \mathsf{Rect}(\mathcal{X})$ is a rectangle and $X \subseteq \mathcal{X}$ is a subset of variables, from Chapter 3. Consider the valuation $\mathbf{a} \in \mathbb{R}^n$ and the m-vector $\langle \eta \rangle = [(\gamma^1, X^1), ..., (\gamma^m, X^m)]$, where, for each $1 \leq j \leq m$, the set $\gamma^j \in \mathsf{Rect}(\mathcal{X})$ is a rectangle and $X^j \subseteq \mathcal{X}$ is a variable set. Then we generate the m-vector of valuations $\langle \mathbf{b} \rangle = [\mathbf{b}^1, ..., \mathbf{b}^m]$ in the following way: for each $1 \leq j \leq m$, we choose a valuation $\mathbf{b}^j \in \mathbb{R}^n$ such that $\mathbf{b}^j \in \mathbf{a}[X^j := \gamma^j]$. That is, for each pair (γ^j, X^j) of the vector $\langle \eta \rangle$, the valuation $\mathbf{b}^j$ must be a member of the set of valuations obtained from $\mathbf{a}$ by resetting the variables in X^j to some value given by the rectangle γ^j. Observe that, for any $1 \leq i, j \leq m$ such that $i \neq j$, it may be the case that $\mathbf{a}[X^i := \gamma^i]$ and $\mathbf{a}[X^j := \gamma^j]$ have a non-empty intersection, and therefore it is possible that $\mathbf{b}^i = \mathbf{b}^j$. Let $\mathsf{Combinations}(\mathbf{a}, \langle \eta \rangle)$ be the set of all such vectors $\langle \mathbf{b} \rangle$ for a given valuation $\mathbf{a}$ and the vector $\langle \eta \rangle$.

In the sequel, we use exclusively vectors of the form of $\langle \eta \rangle$ which consist of post-conditions and variable sets in the support of a distribution. For the distribution p, if $\mathsf{support}(p) = \{(v^1, post^1, X^1), ..., (v^m, post^m, X^m)\}$, then we let the vector $\mathsf{extract}(p) = [(post^1, X^1), ..., (post^m, X^m)]$.

Definition 7.3.1 *The* concurrent probabilistic system $\mathcal{S}_R = (S, \bar{s}, L, \Sigma, Steps)$ *of the probabilistic rectangular automaton* $R = (\mathcal{X}, V, \mathcal{L}, init, inv, flow, prob, \langle pre_v \rangle_{v \in V})$ *is defined as follows:*

- $S \subseteq V \times \mathbb{R}^n$ *is the set of states of* $\mathcal{S}_R$*, defined such that* $(v, \mathbf{a}) \in S$ *if* $\mathbf{a} \in inv(v)$.
- $\bar{s} \in S$ *is the initial state, which is defined such that, for the single location* $\bar{v} \in V$ *satisfying* $init(\bar{v}) \neq \emptyset$*, then* $\bar{s} = (\bar{v}, init(\bar{v}))$ *(recall that* $init(\bar{v})$ *is a singleton, and therefore also a valuation).*
- *For each* $s \in S$*, the atomic propositions assigned to* s *by the labelling function is* $L(s) = \mathcal{L}(\mathsf{discrete}(s))$.
- $\Sigma = \{\theta_p \mid p \in prob(v), v \in V\} \cup \mathbb{R}_{\geq 0}$ *is the event set.*
- *For each state* $(v, \mathbf{a}) \in S$*, let* $Steps(v, \mathbf{a}) = Cts(v, \mathbf{a}) \cup Disc_H(v, \mathbf{a})$ *be the smallest set of event-distribution pairs such that:*
  - *for each duration* $\delta \in \mathbb{R}_{\geq 0}$*, there exists* $(\delta, \mathcal{D}(v, \mathbf{b})) \in Cts(v, \mathbf{a})$ *if and only if either*
    1. $\delta = 0$ *and* $\mathbf{a} = \mathbf{b}$*, or*
    2. $\delta > 0$ *and* $(\mathbf{b} - \mathbf{a})/\delta \in flow(v)$*;*
  - *for each distribution* $p \in prob(v)$*, if* $\mathbf{a} \in pre_v(p)$*, then, for each* $\langle \mathbf{b} \rangle \in \mathsf{Combinations}(\mathbf{a}, \mathsf{extract}(p))$*, there exists the pair* $(\theta_p, \tilde{p}_{\langle \mathbf{b} \rangle}) \in Disc(v, \mathbf{a})$ *such that*

$$\nu_{\langle \mathbf{b} \rangle}(v', \mathbf{c}) = \sum_{\substack{i \in \{1, ..., |\mathsf{support}(p)|\} \\ \& \ \mathbf{c} = \mathbf{b}^i}} p(v', post^i, X^i).$$

As in the case of probabilistic timed automata, we can abstract time from concurrent probabilistic systems of probabilistic rectangular automata by replacing each time duration event with a single event τ.

Definition 7.3.2 *The* time-abstract concurrent probabilistic system $\mathsf{ta}(\mathcal{S}_R) = (S, \bar{s}, L, \Sigma', \mathit{Steps}')$ *of the probabilistic rectangular automaton* $R = (\mathcal{X}, V, \mathcal{L}, \mathit{init}, \mathit{inv}, \mathit{flow}, \mathit{prob}, \langle \mathit{pre}_v \rangle_{v \in V})$ *is defined according to Definition 7.3.1 and as follows.*

Let $\Sigma' = \{\theta_p \mid p \in \mathit{prob}(v), v \in V\} \cup \{\tau\}$*. For every state* $(v, \mathbf{a}) \in S_R$*, let* $\mathit{Steps}'(v, \mathbf{a}) = \mathit{Cts}'(v, \mathbf{a}) \cup \mathit{Disc}(v, \mathbf{a})$*, where there exists* $(\tau, \mathcal{D}(v, \mathbf{b})) \in \mathit{Cts}'(v, \mathbf{a})$ *if there exists* $(\delta, \mathcal{D}(v, \mathbf{b})) \in \mathit{Cts}(v, \mathbf{a})$ *for some* $\delta \in \mathbb{R}_{\geq 0}$*.*

For a state $(v, \mathbf{a}) \in S$, the definition of the continuous transitions in $\mathit{Cts}(v, \mathbf{a})$ is identical to the analogous definition for non-probabilistic rectangular automata (see, for example, [HKPV98]), except that we require them to be made according to Dirac distributions; that is, with probability 1. The definition of the discrete transitions in $\mathit{Disc}(v, \mathbf{a})$ reflects the intuition that a probabilistic rectangular automaton performs a discrete transition in the following manner:

1. by first choosing an enabled distribution *nondeterministically*;

2. next, by selecting a target node and post-condition set *probabilistically*;

3. and finally, choosing a successor state within the post-condition set *nondeterministically.*

It is easy to verify that combining the two nondeterministic choices that comprise the first and third steps of the transition into a single nondeterministic selection, in the manner of Definition 7.3.1, results in an equivalent transition. Note that if the post-condition set of at least one tuple in the support of a distribution p is uncountable, then the number of vectors of the form $\langle \mathbf{b} \rangle$ associated with this set will be uncountable, as will the number of transitions in *Disc* corresponding to p.

7.3.1 Progress for probabilistic rectangular automata

It is not difficult to see that the possibilities of deadlock (a discrete transition is not possible from some point) and timelock (the duration of continuous transitions are forced to converge to a certain bound) could arise in this model. However, as problems of progress for probabilistic rectangular automata are similar to those featured in probabilistic timed automata in Section 4.4, we only briefly address them here. Firstly, we consider dynamic methods for progress in probabilistic rectangular automata, which take into account behaviours and adversaries of the concurrent probabilistic system in question. Note that paths of probabilistic rectangular automata featuring discrete progress can be expressed as those paths with infinitely many discrete transitions; similarly, paths exhibiting time progress can be characterised as letting time pass beyond any bound. The justification for these characterisations is the same as in the case of probabilistic timed automata. Then the definitions of discrete-progress, time-progress and divergent adversaries follows immediately from Definition 4.4.4, Definition 4.4.5 and Definition 4.4.6, respectively.

Note that we can also adapt the two static methods for progress presented in Section 4.4.2; that is, the methods which preclude the situation in which timelock and deadlock could occur together. For the case of non-deadlocking invariants, as the definition of ubound applies to zones, it can also be applied to rectangles. The definition of non-deadlocking invariants (Definition 4.4.8) can then be applied literally to probabilistic rectangular automata. For sufficient deadlock freedom, we observe that, for any location $v \in V$, the definition of $\mathsf{free}(v)$ must be expanded to accommodate the flow condition $\mathit{flow}(v)$. More precisely, the backward projection operator $\swarrow(\gamma)$ can be parameterised by $\mathit{flow}(v)$ to obtain the set of valuations that can reach the rectangle $\gamma \in \mathsf{Rect}(\mathcal{X})$ by letting time elapse, while the first derivatives on the variables with respect to time are subject to the condition $\mathit{flow}(v)$. Similarly, in the definition of sufficient deadlock freedom for probabilistic timed automata (Definition 4.4.9), we must take into account the fact that variables may be reset to a valuation within a rectangle post, rather than just to integer values. However, no new insight is necessary for defining such static methods for progress in the case of probabilistic rectangular automata.

7.4 A probabilistic steam boiler

7.4.1 What is the probabilistic steam boiler?

We now consider the problem of modelling an industrial application, namely that of a *steam boiler*, using probabilistic rectangular automata. The problem of specification and verification of the steam boiler was proposed as a test case for the practicality of formal methods within a non-academic environment by Abrial, Börger and Langmaack [ABL96], and has inspired solutions based on timed automata [KP95, Kri99], hybrid automata [HW96], and a probabilistic guarded command language [MMT98]. Our aim is to combine elements from the latter two solutions to define an appropriate probabilistic hybrid automaton model of the steam boiler, and perform verification of probabilistic reachability and invariance properties on this model. Although the scope of our analysis technique is limited, primarily because computation of the polyhedra used to represent reachable portions of the model's state space is performed manually, we aim to show the feasibility of the application of probabilistic rectangular automata to the formal verification of real-life applications given adequate tool support.

Our approach is to use the steam boiler description of [Abr96] subject to a number of simplifying assumptions. The system consists of a number of physical units, namely a vessel containing an amount of water, a series of water pumps, a series of sensors, and a message transmission system. For simplicity, we restrict the number of pumps

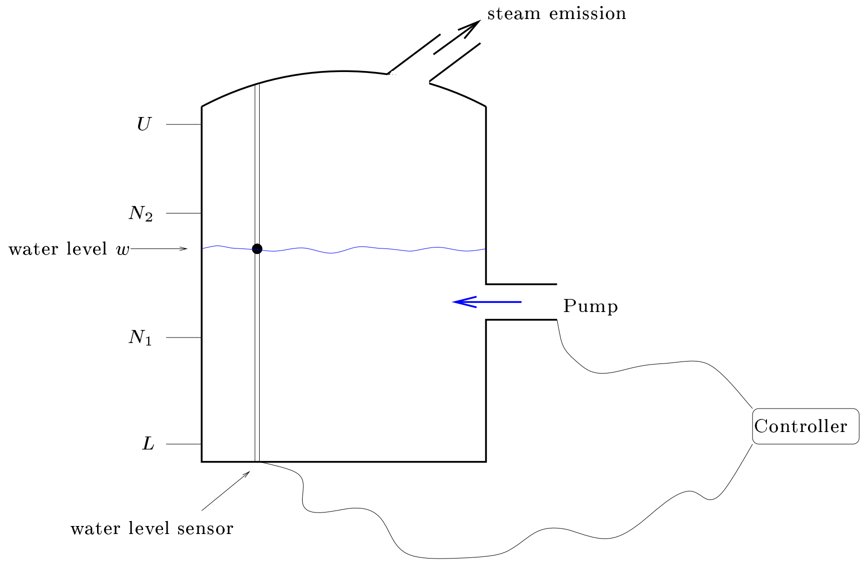


Figure 7.4: A pictorial representation of the probabilistic steam boiler.

from the four suggested in [Abr96] to just one, and assume that this pump can be switched on instantaneously given that the appropriate message has been sent from the controller to the pump ([Abr96] specifies that five seconds are required for water to start pouring into the boiler, which is reflected in the hybrid automaton model of [HW96]). The vessel containing the amount of water is continuously heated in order to produce steam, which can then be used, for example, to drive a turbine. A pictorial representation of the steam boiler is given in Figure 7.4.

A series of quantities is associated with the steam boiler. The constant L denotes the *minimal limit level* of water, with U denoting the corresponding *maximal limit level*. Intuitively, if the water level falls below L, then the fact that the vessel is being heated without containing a sufficient amount of water could mean that permanent damage to the physical units is caused. Similarly, the water level rising above U could correspond to the system being flooded with water. In both cases, the system should enter an "emergency stop" mode in order to avert a catastrophic failure. Furthermore, when the water level remains within the normal operating interval of $[N_1, N_2]$, where N_1 is a constant denoting the *minimal normal level*, and N_2 is a constant denoting the *maximal normal level*, the controller need not intervene. However, if the water level falls below N_1, then the pump is switched on, and, symmetrically, if the water level rises above N_2, then the pump is switched off. Naturally, we require that $L \leq N_1 \leq N_2 \leq U$; that is, the minimum limit water level is less than or equal to the minimum normal water level, and the maximum limit level is greater than the maximum normal level.

The rate of escape of steam is constrained by some maximum M. Although [Abr96] specifies that the acceleration and deceleration of the steam emission is also bounded within an interval, we abstract this characteristic for simplicity (as does the first hybrid automaton model analysed in [HW96]).

Although we assume that the sensors of pump operation and steam emission operate correctly, as in [MMT98], the possibility of a failure in the water level sensor is of vital importance to our analysis. Given the occurrence of such a failure, the controller uses an approximate guess of the actual water level when deciding whether to switch the pump on or off; the estimate may be revised as time elapses according to whether the pump is currently operational or not. Periodically, there is the possibility of the water level sensor being repaired, in which case the system returns to its normal mode of operation. However, as described by [Abr96] and modelled by [MMT98], if the lower bound of the estimate of the water level falls below the lower bound on normal operation N_1, and the upper bound of the estimate exceeds the upper bound on normal operation N_2, then the decision that should be taken by the controller to avoid a catastrophe is not clear; the possibility of the actual water level being below N_1 means that the controller should switch the pump on, and yet the possibility of the level being above N_2 means that the pump should be switched off. Following the precedent of [Abr96, MMT98], we assume that the controller enters a "shutdown" mode in which it no longer engages in communication with the physical units in such a case. We regard this "shutdown" mode as being less catastrophic than the "emergency stop" mode, and therefore it may be tolerated with a higher probability.

Finally, in accordance with [Abr96], communication between the controller and the physical units takes place only once every Δ time units (in [Abr96], Δ is taken to be 5 seconds). However, when communication does occur, the reception of messages from the physical units, their analysis by the controller, and the transmission of messages to the physical units, are all taken to be instantaneous.

7.4.2 Modelling the probabilistic steam boiler

A probabilistic rectangular automaton R_{sb} to model the steam boiler system described above is given in Figure 7.5. In order to explain why R_{sb} is an appropriate model for the system, we first explain the role of its four continuous variables, then proceed on to the locations and distributions.

The variable w denotes the water level, t is a clock, g_l represents the lower bound on the current guess on the water level, and g_u represents the corresponding upper bound. Of these, the variable w is subject to the most involved continuous dynamics, which we now explain. Intuitively, the value of the water level is affected by two aspects of the behaviour of the physical units: that of the pump, and that of the steam. When the pump is off, the water level is subject to continuous change reflecting steam emission only; as with the first model analysed by [HW96], we express the rate of change of the steam emission volume as the interval $[0, E]$ for some positive

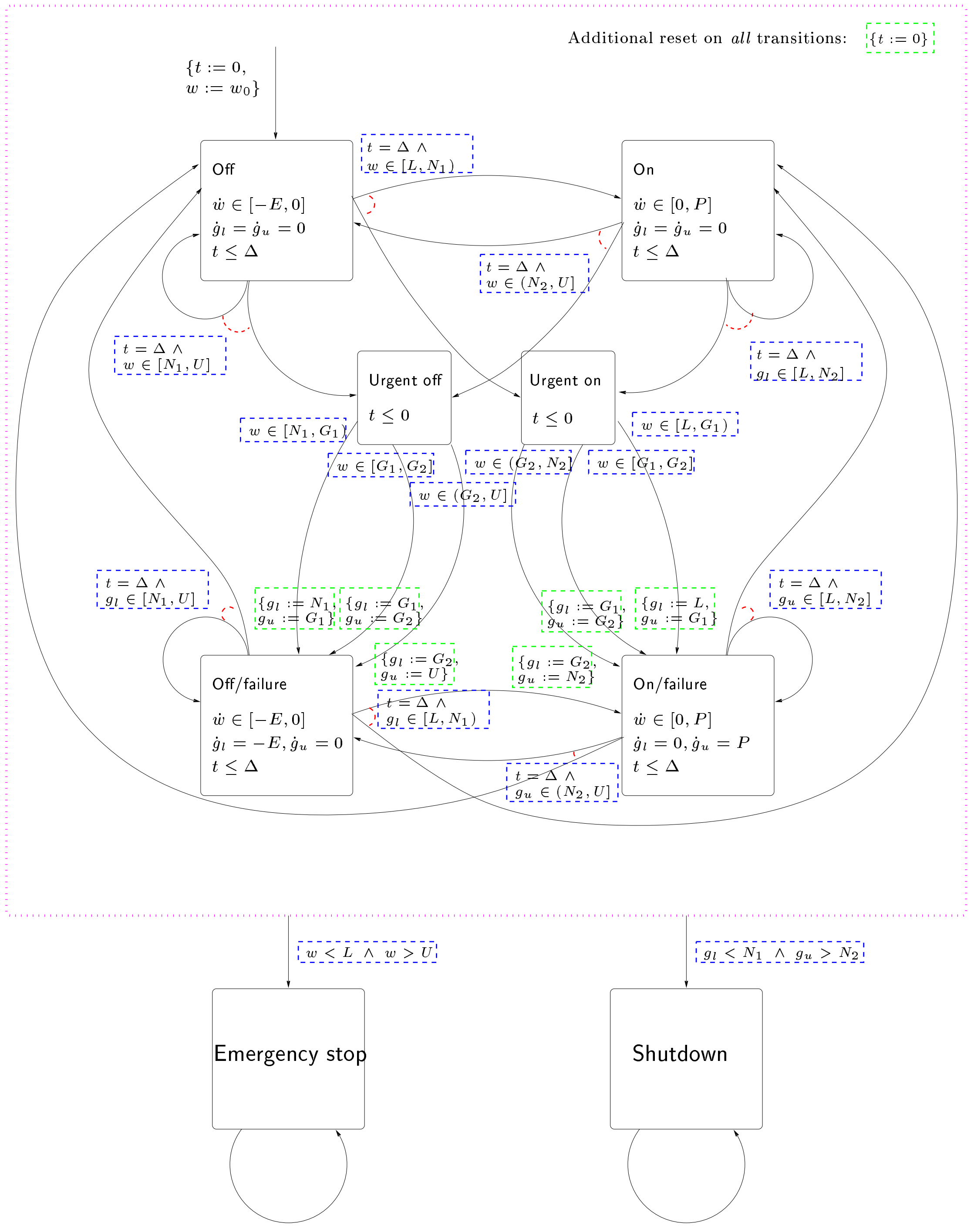

Additional reset on *all* transitions: $\{t := 0\}$
$\{t := 0, w := w_0\}$
Off
$\dot{w} \in [-E, 0]$
$\dot{g}_l = \dot{g}_u = 0$
$t \leq \Delta$
On
$\dot{w} \in [0, P]$
$\dot{g}_l = \dot{g}_u = 0$
$t \leq \Delta$
$t = \Delta \wedge w \in [L, N_1)$
$t = \Delta \wedge w \in (N_2, U]$
$t = \Delta \wedge w \in [N_1, U]$
$t = \Delta \wedge g_l \in [L, N_2]$
Urgent off
$t \leq 0$
Urgent on
$t \leq 0$
$w \in [N_1, G_1)$
$w \in [L, G_1)$
$w \in [G_1, G_2]$
$w \in (G_2, N_2]$
$w \in [G_1, G_2]$
$w \in (G_2, U]$
$t = \Delta \wedge g_l \in [N_1, U]$
$\{g_l := N_1, g_u := G_1\}$
$\{g_l := G_1, g_u := G_2\}$
$\{g_l := G_1, g_u := G_2\}$
$\{g_l := L, g_u := G_1\}$
$t = \Delta \wedge g_u \in [L, N_2]$
$\{g_l := G_2, g_u := U\}$
$\{g_l := G_2, g_u := N_2\}$
$t = \Delta \wedge g_l \in [L, N_1)$
Off/failure
$\dot{w} \in [-E, 0]$
$\dot{g}_l = -E, \dot{g}_u = 0$
$t \leq \Delta$
On/failure
$\dot{w} \in [0, P]$
$\dot{g}_l = 0, \dot{g}_u = P$
$t \leq \Delta$
$t = \Delta \wedge g_u \in (N_2, U]$
$w < L \wedge w > U$
$g_l < N_1 \wedge g_u > N_2$
Emergency stop
Shutdown

Figure 7.5: A probabilistic hybrid automaton H_{sb} for a steam boiler.

integer constant E (for simplicity, we do not consider the more accurate linear hybrid automaton approximation given in [HW96]). Therefore, in locations corresponding to the deactivation of the pump, the water level will be subject to the rectangular flow inclusion $\dot{w} = [-E, 0]$. In contrast, when the pump is on, the water level is subject to change due to both steam emission and the flow of water into the tank. We let the rectangular flow inclusion associated with the appropriate locations be $\dot{w} = [0, P]$ for some positive integer constant P, the lower bound of which reflects the assumption that the steam emission could negate completely the effect of water being pumped into the tank (that is, the water level may remain constant).

The eight locations can be split into four categories, each comprising two locations. The first three of these categories consist of one location corresponding to the pump being off (the location Off), the other to it being switched on (the location On). The second consists of "urgent" locations, Urgent off and Urgent on, in which no time is permitted to elapse. The purpose of the two locations is to ease the graphical notation of the probabilistic hybrid automaton. Intuitively, they correspond to the controller making estimates of the water level (a process which, naturally, is instantaneous). The third category comprises the locations Off/failure and On/failure, which correspond to the operation of the steam boiler when the water level sensor has malfunctioned. Finally, the fourth category corresponds to the terminating behaviour of the steam boiler. The location Emergency stop is entered if the water level falls below the minimum limit value or exceeds the maximum limit value. The location Shutdown is entered if both the lower and upper bounds on the estimate of the water level contain the bounds on the water level required for normal operation.

Locations Off and On. The first category captures the behaviour of the steam boiler during normal, fault-free operation. When control resides in the location Off (which is also the initial location of R_{sb}) the water level decreases by between 0 and E litres per time unit; therefore, $\dot{w} \in [-E, 0]$. As the system has an accurate reading of the water level, it does not need to use an estimate in order to decide how to control the pump. Therefore, the continuous behaviour of the variables g_l and g_u may be arbitrary; we let them be constant (denoted by $\dot{g_l} = \dot{g_u} = 0$) for simplicity. Similarly, the location On corresponds to the pump being on during fault-free operation, and therefore the water level is defined to increase at any rate between 0 and P litres per time unit; hence, the condition of this location includes the constraint on $\dot{w}$ denoted by $\dot{w} \in [0, P]$.

Both locations Off and On have the invariant condition $t \leq \Delta$; therefore, control must leave either of these locations if the value of the clock t is equal to Δ. Each location is associated with two distributions. First, consider the location Off and the distribution with the enabling condition $t = \Delta \wedge w \in [L, N_1)$. Intuitively, this distribution is available for choice when the value of the clock t is equal to Δ, and the water level is greater than or equal to the minimum limit level L and strictly below the minimum normal operating level N_1. This corresponds to the case in which the water level has fallen to a level below the normal operating bounds, and therefore the controller sends a message to the pump to switch on. Therefore, both of the edges in the support of this distribution lead to locations in which the pump is to be activated, namely On and Urgent on (the latter of which corresponds to a water level sensor failure).

The other distribution is enabled when $t = \Delta$ and $w \in [N_1, U]$. Recall that when the clock t reaches Δ, the physical plant and the controller communicate. However, if the water level w is equal to or above its minimum normal level N_1, or equal to or below its maximum limit level U, then the controller will decide that no intervention is necessary; therefore, the pump can remain switched off. Hence, one edge in the support of this distribution leads to the location Urgent off, and the other is a self-loop to Off. Note that the clock t is set to 0 when *any* of the edges of the probabilistic rectangular automaton are traversed, apart from those leading to terminating locations (as denoted in the upper-right corner of the box enclosing the non-terminating locations and their edges).

Observe that the behaviour of the system in the location On is, broadly speaking, symmetric to that in Off. The water level increases at a rate between 0 and P units per time unit, and the two distributions associated with this location are such that one leads to Off and Urgent on, whereas the other leads to Urgent on and loops back to On. The enabling condition of the former distribution is $t = \Delta \wedge w \in (N_2, U]$; therefore, the controller requires that the pump is turned off if the water level exceeds the maximum limit value N_2. The latter distribution is enabled when $t = \Delta \wedge w \in [L, N_2]$ is satisfied; that is, the water level is low enough for the controller to require that the pump remains activated.

Locations Urgent off and Urgent on. Recall that these two locations are defined so that no time can elapse while control resides in either location. This is enforced by the invariant conditions $t \leq 0$ of both locations (observe that the value of the clock t is 0 when either location is entered). Because no time can elapse, flow conditions are irrelevant, and we therefore omit them from the graphical representation. All of the outgoing distributions from both locations Urgent off and Urgent on contain only one edge in their support, and therefore, depending on which distribution is selected for choice, the target state is reached with probability 1. First consider the location Urgent off, and observe that it can only be entered if the water level is between N_1 and U. The three distributions available in Urgent off have mutually exclusive enabling conditions which depend only on the water level. Furthermore, all of the distributions lead to the same location Off/failure, differing only in the way in which they reset the "guess" variables, g_l and g_u. That is, if the water level is in the interval $[N_1, G_1)$, for some integer $N_1 < G_1 < N_2$, then the lower bound on the guess, as represented by the variable g_l, is set to N_1, whereas the upper bound, represented by g_u, is set to

G_1. Similarly, if $w \in [G_1, G_2]$, for some integer $G_1 < G_2 < N_2$, then g_l and g_u are set to G_1 and G_2 respectively, and if $w \in (G_2, U]$, then g_l and g_u are set to G_2 and U respectively. Intuitively, the integer constants G_1, G_2 represent boundaries on the real water level which the controller can use when making its guess. The behaviour in the location Urgent on is analogous (naturally, all of the distributions lead to the location On/failure), and the reader can check that the enabling and post-conditions of these distributions are appropriate.

Locations Off/failure and On/failure. Naturally, these two locations correspond to the case in which the water level sensor has failed. The behaviour of the probabilistic rectangular automaton is similar to that in the locations Off and On, except for the following details. Firstly, as the controller is now maintaining an estimate of the water level, the flow conditions of the variables representing the bounds on the guess of the water level, g_l and g_u, are altered to take into account the fact that the real water level may change as time elapses. That is, in the location Off/failure, in which the real water level is subject to the flow condition $\dot{w} \in [-E, 0]$, the lower bound on the estimate is subject to the flow condition $\dot{g_l} = -E$, whereas the upper bound is subject to $\dot{g_u} = 0$. The continuous dynamics for the location On/failure are similar in that the flow conditions for the real water level, $\dot{w} \in [0, P]$, is reflected in the conditions $\dot{g_l} = 0$, $\dot{g_u} = P$ for the lower and upper bounds on the estimate, respectively. As in the location Off, two distributions are associated with the location Off/failure; the first is enabled when the lower estimate on the water level, g_l (rather than the actual water level, w) is in the interval $[L, N_1)$, and leads to the locations On/failure and On; naturally, the selection of this distribution corresponds to the controller choosing to activate the pump on the basis of the lower bound on the estimate of the water level. The transition to the location On corresponds to a repair in the water level sensor, whereas the transition to On/failure corresponds to continued sensor failure. The second distribution is enabled when the lower bound on the water level estimate is equal to or above the minimal normal level N_1 and less than or equal to the maximal limit level U; if this distribution is taken, there is a probability of making a transition to Off (corresponding to a sensor repair), or a self-loop back to Off/failure.

The behaviour of the model in the location On/failure is analogous to that in Off/failure and On. We observe the following important fact: in the location On/failure, the controller will make decisions concerning whether to deactivate the pump based on the *upper* bound on the estimate of the water level, g_u (rather than on the corresponding lower bound, g_l, in Off/failure, or on the actual water level, w, in Off and On). The enabling conditions of the two distributions associated with On/failure follow from this fact and the similar enabling conditions of the distributions associated with the location On; that is, one distribution is enabled when $t = \Delta \wedge g_u \in (N_2, U]$, whereas the other distribution is enabled when $t = \Delta \wedge g_u \in [L, N_2]$. Naturally, the former distribution corresponds to the controller choosing to deactivate the pump, and therefore is defined over edges leading to the locations Off/failure and Off, whereas the latter distribution corresponds to the controller choosing to take no action, and therefore is defined over an edge leading to the location On and a self-loop edge.

Locations Emergency stop and Shutdown. Each of the six locations described above has a number of common features. Firstly, we require that, if the real water level falls below the lower limit L or exceed the upper limit U, then control can pass to the terminal location Emergency stop. We ease the graphical notation by enclosing the six locations in a dotted box, and draw a single edge from the box to the location Emergency stop with the enabling condition $w < l \vee w > U$; this represents the fact that there is a distribution over a single edge of each of the six locations to Emergency stop.

Similarly, each of the six locations should also be able to make a transition to a terminal location Shutdown if the lower bound on the estimate of the water level is below the lower normal water level at the same time as the upper bound on the estimate is above the upper normal level. The same style of graphical notation as used for the Emergency stop location is used here; that is, the distribution over a single edge leading to the location Shutdown, which is enabled when the condition $g_l < N_1 \wedge g_u > N_2$ is satisfied, is represented by the edge from the dotted box to this location.

We choose not to specify the exact value of the probabilities of the transitions at this point (unless they occur with probability 1). However, we note that it may be realistic for the values of the probabilities corresponding to normal, non-malfunctioning behaviour to be substantially higher than those corresponding to the occurrence of faults. For example, the transition from the location Off to the location On is likely to have a higher probability than that from Off to Urgent on in a realistic system.

7.4.3 An example run of the steam boiler

An example run of the steam boiler system is shown in Figure 7.6. The values of the water level w, and the lower and upper bounds on the estimate of the water level, g_l and g_u, are represented on the y-axis, whereas the x-axis represents the global time of system execution. As the variable t is a clock, the values that it takes are denoted on the x-axis in a natural way. Furthermore, we also annotate this axis with labels denoting the location in which control resides at the appropriate times during the execution of the system.

We assume that the system constants are such that $L < N_1 < G_1 < G_2 < N_2 < U$, and that the initial water level of the boiler, denoted by w_0, is some point in the interval (G_1, G_2). Also, the value of the clock t is 0 initially. When the system commences operation, the pump is switched off (control resides in the location Off), and the water level either stays constant or falls at some rate no more than E. The system

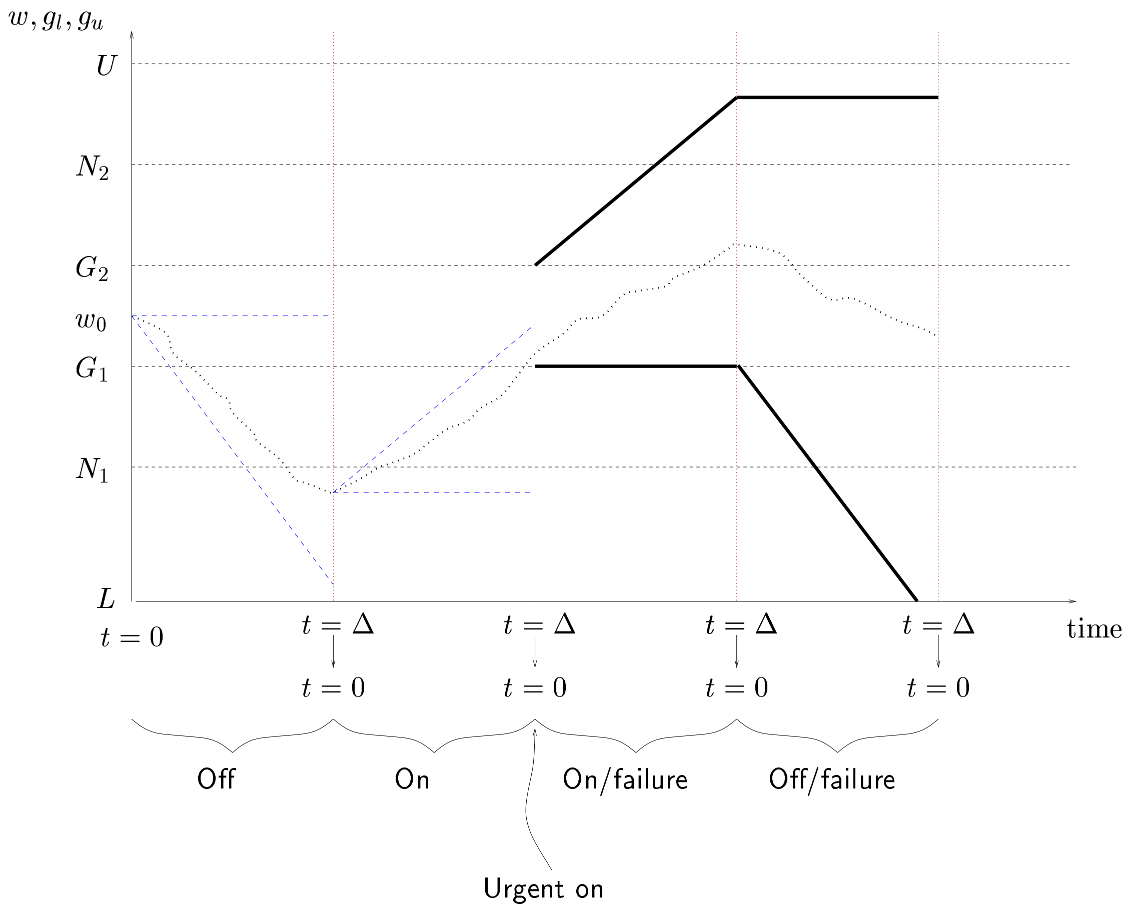


Figure 7.6: An example run of the probabilistic rectangular automaton H_{sb}.

must remain in this mode until the clock t increases to equal Δ, at which point the controller communicates with the physical units. Say the water level has fallen below the minimal normal level N_1, as shown in Figure 7.6. Then the controller decides to switch the pump on. Consider the case in which a failure does not occur at this point; therefore, control passes successfully to the location On, and the clock t is reset to 0. The water level now remains constant or rises no more than P litres per time unit. Consider the case in which it rises to between G_1 and G_2. When $t = \Delta$ again, and the controller can communicate with the physical units, the level is not high enough to warrant intervention, so the pump remains switched on. However, say the water level sensor fails at this point. Therefore, control passes to the location Urgent on, with the clock t being reset to 0. As the water level is between the values G_1 and G_2, the lower bound on the estimate g_l is set to G_1, while the upper bound g_u is set to G_2, and control passes instantaneously to the location On/failure. Then, as time passes, the real water level will increase at a rate in the interval $[0, P]$, while g_u will increase at rate P, and g_l will remain constant. Diagrammatically, the upper bound on the guess g_u is represented by the uppermost (upward sloping) bold line in this segment of the figure, with the corresponding lower bound being the bold horizontal line; the real value of the water level lies between these two lines.

When t reaches Δ, the controller will now decide to deactivate the pump, as the upper bound on the estimate, the variable g_u, is above the maximum normal limit. Say the water level sensor is not repaired at this point; therefore, control of the system passes to the location Off/failure, and t is again reset. The upper bound on the estimate now remains constant, the lower bound decreases at rate E, and the real water level decreases at any rate in the interval $[0, E]$. However, before the clock t increases to reach the value Δ, the system is in the untenable situation in which the upper bound on the estimate exceeds the maximum normal level, and the lower bound on the estimate falls below the minimum normal level. Therefore, control of the system enters the location Shutdown.

Chapter 8

Translations for probabilistic hybrid automata

8.1 Introduction

The next two chapters present strategies for the model checking of probabilistic rectangular automata against probabilistic properties. The first method, dealt with in this chapter, concerns the translation of various restricted subclasses of probabilistic rectangular automata to probabilistic timed automata, for which the verification methods of Chapters 5 and 6 can be used. This strategy involves modification of the analogous results presented in the non-probabilistic hybrid automaton literature [OSY94, HKPV98] to our context, which primarily involves showing that our translations are *probabilistic* simulations and bisimulations in the manner of [JL91, LS91, SL95], as defined in Definition 3.2.15 and Definition 3.2.16.

To explain our intuition, recall the probabilistic hybrid system of a faulty production line, as presented in Section 7.2. In a system as simple as the production line, it is not difficult to see that mechanisms to represent timing issues could be used indirectly to model other continuous aspects of the hybrid system, such as the position along the conveyor belt. For example, when the item is placed on the first segment, it will take between 1 and $1\frac{1}{2}$ minutes to reach the end of the segment. This idea can be used as a basis of a *translation* from a (probabilistic) rectangular automaton to a (probabilistic) timed automaton. As we have presented model checking methods for probabilistic timed automata against properties expressed in a probabilistic logic in Chapter 5 and Chapter 6, then, provided that the translation methods preserve the properties of the probabilistic logic (either in an *exact* or *sufficient* way), we also have a model checking method for probabilistic rectangular automata against probabilistic logic.

However, these results are subject to powerful side-conditions concerning the type of probabilistic rectangular automata to which they can be applied. In particular, the models must be *initialised*; recall from Section 7.2.5 that initialisation requires that the value of any variable must be reset (either nondeterministically within a range of values, or deterministically to a single value) when the requirements on the continuous dynamics of the variable change.

In each of the sections of this chapter, we consider a translation method for a class of probabilistic rectangular automata. We proceed by detailing the translation from a probabilistic stopwatch automaton to a timed model in the next section, then show how a probabilistic multisingular automaton can be translated to a probabilistic stopwatch automaton in Section 8.3. Finally, in Section 8.4, we show how to translate a probabilistic rectangular automaton into a multisingular model. Note that parts of this chapter have appeared in [Spr99, Spr00].

8.2 Translation: stopwatch to timed

As noted in [HKPV98] in the non-probabilistic setting, an initialised stopwatch automaton can be transformed into a bisimilar timed automaton. The intuition underlying this result is that the values of stopped variables (those with $\dot{x} = 0$) can be encoded into the locations of the timed automaton. Because the assumptions of initialisation and deterministic jumps (the latter assumption is integral to the definition of stopwatch automata) mean that there are only a finite number of integer values that stopped variables can take, such an encoding always results in a finite number of locations. We now extend this result to the probabilistic case to derive a method for translating initialised probabilistic stopwatch automata to probabilistic timed automata. Recall from Chapter 5 that we have a PBTL model checking method for probabilistic timed automata, as PBTL is the subset of PTCTL without reset quantifiers and zones as subformulae. As bisimulation preserves PBTL formulae (see Theorem 3.3.6), it then follows that we are able to use the translation method below to model check initialised probabilistic stopwatch automata against PBTL formulae.

8.2.1 Definition of the translation: stopwatch to timed

From a initialised probabilistic stopwatch automaton

$$W = (\mathcal{X}, V^W, \mathcal{L}^W, \mathit{init}^W, \mathit{inv}^W, \mathit{flow}^W, \mathit{prob}^W, \langle \mathit{pre}_v^W \rangle_{v \in V^W}) ,$$

we construct the probabilistic timed automaton

$$T_W = (\mathcal{X}, V^{T_W}, \mathcal{L}^{T_W}, \mathit{init}^{T_W}, \mathit{inv}^{T_W}, \mathit{prob}^{T_W}, \langle \mathit{pre}_{(v,f)}^{T_W} \rangle_{(v,f) \in V^{T_W}})$$

in the following way. Let $\mathcal{X} = \{x_1, ..., x_n\}$ as usual.

Locations. Let $\mathcal{K}_W$ be the set of integer constants used in the definition of W, and let $\mathcal{K}_\perp = \mathcal{K}_W \cup \{\perp\}$. Then $V^{T_W} = V^W \times \mathcal{K}_\perp^{\mathcal{X}}$; therefore, each location of T_W

consists of a location of W and a function $f : \mathcal{X} \to \mathcal{K}_\perp$. For a given location of W, say $v \in V^W$, a function f of $(v, f) \in V^{T_W}$ will have the following form: for each $x_i \in \mathcal{X}$, if $\mathit{flow}^W(v)_i$ is $\dot{x}_i = 1$, then $f(x_i) = \perp$; otherwise, $f(x_i) \in \mathcal{K}_W$.

Labelling condition. For each location $(v, f) \in V^{T_W}$, let $\mathcal{L}^{T_W}(v, f) = \mathcal{L}^W(v)$.

Initial condition. For all locations $(v, f) \in V^{T_W}$, if, for each $x_i \in \mathcal{X}$, either $f(x_i) = \perp$ or $f(x_i) = \mathit{init}^W(v)_i$, then let $\mathit{init}^{T_W}(v, f) = \mathit{init}^W(v)$, otherwise let $\mathit{init}^{T_W}(v, f) = \emptyset$.

Invariant conditions. For all locations $(v, f) \in V^{T_W}$, if $f(x_i) = \perp$ then let $\mathit{inv}^{T_W}(v, f) = \mathit{inv}^W(v)$. Otherwise, if $f(x_i) \in \mathit{inv}^W(v)_i$ then let $\mathit{inv}^{T_W}(v, f)_i = \mathbb{R}_{\geq 0}$, and if $f(x_i) \notin \mathit{inv}^W(v)_i$ then let $\mathit{inv}^{T_W}(v, f)_i = \emptyset$.

Probability distributions. Recall that, for each location $v \in V^W$, $\mathit{prob}^W(v) = \{p_v^1, ..., p_v^l\}$ for some finite $l \geq 1$. Then, for each location of the form $(v, f) \in V^{T_W}$, let $\mathit{prob}^{T_W}(v, f) = \{p^1_{(v,f)}, ..., p^l_{(v,f)}\}$, where, for each $1 \leq j \leq l$, the distribution $p^j_{(v,f)}$ is derived from p^j_v in the following manner. For each tuple $(v', \mathit{post}, X) \in V^W \times \mathbb{N}^n \times 2^{\mathcal{X}}$, there exists $(v', g) \in V^{T_W}$ such that $p^j_{(v,f)}((v', g), \mathit{post}, X) = p^j_v(v', \mathit{post}, X)$, and, for every variable $x_i \in \mathcal{X}$, either:

1. $\mathit{flow}^W(v')_i = [1, 1]$ and $g(x_i) = \perp$,
2. $\mathit{flow}^W(v)_i = [1, 1]$, $\mathit{flow}^W(v')_i = [0, 0]$ and $g(x_i) = \mathit{post}_i$,
3. $\mathit{flow}^W(v)_i = [0, 0]$, $\mathit{flow}^W(v')_i = [0, 0]$, $x_i \notin X$ and $g(x_i) = f(x_i)$,
4. $\mathit{flow}^W(v)_i = [0, 0]$, $\mathit{flow}^W(v')_i = [0, 0]$, $x_i \in X$ and $g(x_i) = \mathit{post}_i$.

Let $p^j_{(v,f)}((v'', g'), \mathit{post}, X) = 0$ for all other $(v'', g') \in V^{T_W}$.

Pre-conditions. For all locations $(v, f) \in V^{T_W}$, for all $1 \leq j \leq |\mathit{prob}^{T_W}(v, f)|$, and for all variables $x_i \in \mathcal{X}$, if $f(x_i) = \perp$, then $\mathit{pre}^{T_W}_{(v,f)}(p^j_{(v,f)})_i = \mathit{pre}^W_{(v,f)}(p^j_v)_i$. Otherwise we obtain $\mathit{pre}^{T_W}_{(v,f)}(p^j_{(v,f)})_i$ by substituting the value of $f(x_i)$ for x_i in $\mathit{pre}^W_v(p^j_v)_i$; that is, if $f(x_i) \in \mathit{pre}^W_v(p^j_v)_i$, then let $\mathit{pre}^{T_W}_{(v,f)}(p^j_{(v,f)})_i = \mathbb{R}_{\geq 0}$, otherwise let $\mathit{pre}^{T_W}_{(v,f)}(p^j_{(v,f)})_i = \emptyset$.

This concludes the definition of the probabilistic timed automaton T_W. Note that, equivalently, the locations of T_W can be obtained by equipping W with extra variables which take values over a *finite* set of possibilities. More precisely, there will be $|\mathcal{X}|$ such extra variables, which take values from the set $\mathcal{K}_\perp$, and which remain constant while control remains in a particular mode. The facility for reasoning about such variables exists in the real-time model checking tools UPPAAL [LPY97a], and OPEN-KRONOS (an adaptation of KRONOS) [Tri98].

8.2.2 Equivalence: stopwatch and timed

Observe that the concurrent probabilistic systems, $\mathcal{S}_W = (S_W, \bar{s}_W, L_W, \Sigma_W, \mathit{Steps}_W)$ and $\mathcal{S}_{T_W} = (S_{T_W}, \bar{s}_{T_W}, L_{T_W}, \Sigma_{T_W}, \mathit{Steps}_{T_W})$ of S and T_S respectively can be obtained by straightforward use of Definition 7.3.1. Following the precedent of [HKPV98], we introduce the function $\mathsf{t2w}$, which maps states of the concurrent probabilistic system $\mathcal{S}_{T_W}$ to states of the concurrent probabilistic system $\mathcal{S}_W$. More precisely, $\mathsf{t2w} : S_{T_W} \to S_W$ is a surjective function relating states of the underlying model $\mathcal{S}_{T_W}$ of the probabilistic timed automaton T_W, to those of the underlying model $\mathcal{S}_W$ of the initialised probabilistic stopwatch automaton W. We then show that an associated equivalence relation, $\approx_{\mathsf{t2w}}$, on the state set $S_W \cup S_{T_W}$, is a probabilistic bisimulation in the manner of Definition 3.2.16.

Definition 8.2.1 (cf. [HKPV98]) *Let W be an initialised probabilistic stopwatch automaton, let T_W be a probabilistic timed automaton derived from W in as in Section 8.2, and let $\mathcal{S}_W$ and $\mathcal{S}_{T_W}$ be their associated concurrent probabilistic systems. Then let $\mathsf{t2w} : S_{T_W} \to S_W$ be the function defined such that, for each $((v,f),\mathbf{a}) \in S_{T_W}$, we have $\mathsf{t2w}((v,f),\mathbf{a}) = (v,\mathbf{b})$, where $\mathbf{a}_i = \mathbf{b}_i$ if $f(x_i) = \bot$, and $\mathbf{b}_i = f(x_i)$ otherwise, for each variable $x_i \in \mathcal{X}$.*

The relation $\approx_{\mathsf{t2w}}$ is an equivalence on $S_{T_W} \cup S_W$ such that $s \approx_{\mathsf{t2w}} s'$ if and only if $\mathsf{t2w}(s) = s'$ for $s \in S_{T_W}$ and $s' \in S_W$.

Proposition 8.2.2 *Let W be an initialised probabilistic stopwatch automaton, and let T_W be the probabilistic timed automaton derived from W using the translation method of Section 8.2. Then the equivalence $\approx_{\mathsf{t2w}}$ is a bisimulation between $\mathcal{S}_W$ and $\mathcal{S}_{T_W}$.*

Proof. To prove the proposition, we first show that, for states $((v,f),\mathbf{a}) \in S_{T_W}$ and $(v,\mathbf{b}) \in S_W$ such that $((v,f),\mathbf{a}) \approx_{\mathsf{t2w}} (v,\mathbf{b})$, then $((v,f),\mathbf{a}) \simeq (v,\mathbf{b})$. We deal with the conditions of the definition of bisimulation (Definition 3.2.16) in turn.

Firstly, we require that $L_{T_W}((v,f),\mathbf{a}) = L_W(v,\mathbf{b})$. From the construction of T_W, we have $\mathcal{L}^{T_W}(v,f) = \mathcal{L}^W(v)$. From the construction of the concurrent probabilistic systems $\mathcal{S}_{T_W}$ and $\mathcal{S}_W$, we have $L_{T_W}((v,f),\mathbf{a}) = \mathcal{L}^{T_W}(v,f)$ and $L_W(v,\mathbf{b}) = \mathcal{L}^W(v)$, respectively. Therefore, we have $L_{T_W}((v,f),\mathbf{a}) = L_W(v,\mathbf{b})$.

Secondly, we require that there exists the transition $((v,f),\mathbf{a}) \xrightarrow{\sigma,\tilde{p}}$ if and only if there exists the transition $(v,\mathbf{b}) \xrightarrow{\sigma,\tilde{p}'}$, where $\tilde{p} \approx_{\mathsf{t2w}} \tilde{p}'$. There are two cases, depending on whether a time transition or a discrete transition is made.

Time transition. We show that the time transition $((v,f),\mathbf{a}) \xrightarrow{\delta} ((v,f),\mathbf{a}+\delta)$ exists if and only if the time transition $(v,\mathbf{b}) \xrightarrow{\delta} (v,\mathbf{c})$ exists, where $((v,f),\mathbf{a}+\delta) \approx_{\mathsf{t2w}} (v,\mathbf{c})$, also exists.

Note that $\mathbf{b} \in \mathit{inv}^W(v)$ and $\mathbf{a} \in \mathit{inv}^{T_W}(v,f)$. We proceed by showing that, for each variable $x_i \in \mathcal{X}$, we have

- $\mathbf{c}_i \in \mathit{inv}^W(v)_i$ if and only if $\mathbf{a}_i + \delta \in \mathit{inv}^{T_W}(v,f)_i$, and
- $\mathbf{c}_i = \mathbf{a}_i + \delta$ if $\mathit{flow}^W(v)_i = [1,1]$, and $\mathbf{c}_i = f(x_i)$ if $\mathit{flow}^W(v)_i = [0,0]$.

We consider the following two cases, which depend on the flow condition of the location v of W.

Case: $\mathit{flow}^W(v)_i = [1,1]$. Then, by the construction of T_W, we have $\mathit{inv}^W(v)_i = \mathit{inv}^{T_W}(v)_i$, and, by $((v,f),\mathbf{a}) \approx_{\mathsf{t2w}} (v,\mathbf{b})$, we have $\mathbf{a}_i = \mathbf{b}_i$. Furthermore, the condition $\mathit{flow}^W(v)_i = [1,1]$ implies that $\mathbf{c}_i = \mathbf{b}_i + \delta$, and therefore $\mathbf{c}_i = \mathbf{a}_i + \delta$. Hence $\mathbf{c}_i = \mathbf{a}_i + \delta \in \mathit{inv}^W(v)_i$ if and only if $\mathbf{a}_i + \delta \in \mathit{inv}^{T_W}(v)_i$. By the convexity of the sets $\mathit{inv}^W(v)_i$ and $\mathit{inv}^{T_W}(v)_i$, we have $\mathbf{a}_i + \delta' \in \mathit{inv}^W(v)_i$ and $\mathbf{a}_i + \delta' \in \mathit{inv}^{T_W}(v)_i$ for any $0 \leq \delta' \leq \delta'$.

Case: $\mathit{flow}^W(v)_i = [0,0]$. From $((v,f),\mathbf{a}) \approx_{\mathsf{t2w}} (v,\mathbf{b})$ we have $\mathbf{b}_i = f(x_i)$. For $(v,\mathbf{b})$ to be an admissible state, we have $\mathbf{b}_i \in \mathit{inv}^W(v)_i$. Then, from this fact and the construction of T_W, we have $\mathit{inv}^{T_W}(v,f)_i = \mathbb{R}_{\geq 0}$. Hence, whatever the value of $\delta \in \mathbb{R}_{\geq 0}$, we have $\mathbf{a}_i + \delta \in \mathit{inv}^{T_W}(v,f)_i$.

As $\mathit{flow}^W(v)_i = [0,0]$, it must be the case that $\mathbf{c}_i = \mathbf{b}_i$, and from $\mathbf{b}_i = f(x_i)$, we have $\mathbf{c}_i = f(x_i)$. Therefore, the valuations of the variable x_i are such that the appropriate condition of $\approx_{\mathsf{t2w}}$ is satisfied.

Repeating this process for all variables $x_i \in \mathcal{X}$ implies that both of the transitions $((v,f),\mathbf{a}) \xrightarrow{\delta} ((v,f),\mathbf{a}+\delta)$ and $(v,\mathbf{b}) \xrightarrow{\delta} (v,\mathbf{c})$ exist, and that $((v,f),\mathbf{a}+\delta) \approx_{\mathsf{t2w}} (v,\mathbf{c})$.

Discrete transition. We must also consider the case in which $((v,f),\mathbf{a}) \xrightarrow{\sigma,\tilde{p}}$ and $(v,\mathbf{b}) \xrightarrow{\sigma,\tilde{p}'}$ are discrete transitions. We proceed by showing that, for the set of distributions that can be nondeterministically chosen in $(v,\mathbf{b})$, the set of distributions available for choice in $((v,f),\mathbf{a})$ are exactly those that are derived from the former set in the construction method for T_W. That is, if $\mathit{prob}^W(v) = \{p_v^1, ..., p_v^l\}$ and $\mathit{prob}^{T_W}(v,f) = \{p_{(v,f)}^1, ..., p_{(v,f)}^l\}$, then, for all $1 \leq j \leq l$, if the valuation $\mathbf{b}$ satisfies $\mathit{pre}_v^W(p_v^j)$, then the valuation $\mathbf{a}$ satisfies $\mathit{pre}_{(v,f)}^{T_S}(p_{(v,f)}^j)$. This fact follows immediately from the definitions of $\mathsf{t2w}$ and the pre-conditions of the distributions of T_S.

Secondly, take a particular $1 \leq j \leq l$. Recall that bisimulation can be characterised in an alternative way which requires that transitions are made with distributions that assign the same total probability to equivalence classes (see Proposition 3.2.17). We now show that the distribution $\tilde{p}_v$ of $\mathcal{S}_W$, which is derived from the distribution of W denoted by p_v^j, and the distribution $\tilde{p}_{(v,f)}$ of $\mathcal{S}_{T_W}$, which is derived from the distribution of T_W denoted by $p_{(v,f)}^j$, are distributions such that $\tilde{p}_v \approx_{\mathsf{t2w}} \tilde{p}_{(v,f)}$. Take a tuple $(v', \mathit{post}, X) \in V^W \times \mathbb{N}^n \times 2^{\mathcal{X}}$, with an associated tuple $((v',g), \mathit{post}, X) \in V^{T_W} \times \mathbb{N}^n \times 2^{\mathcal{X}}$ such that $p_{(v,f)}^j((v',g), \mathit{post}, X) = p_v^j(v', \mathit{post}, X)$ (by the definition of the translation from W to T_W, such a tuple will exist). Then:

- the state $(v', \mathbf{b}')$, which is reached from $(v,\mathbf{b})$ via nondeterministic choice of p_v^j and then probabilistic choice of (v', post, X), and

- the state $((v', g), \mathbf{a}')$, reached from $((v, f), \mathbf{a})$ via nondeterministic choice of $p^j_{(v,f)}$ and then probabilistic choice of $((v', g), post, X)$,

are such that $((v', g), \mathbf{a}') \approx_{\mathsf{t2w}} (v', \mathbf{b}')$. To see this, note that it follows from the definition of $\mathsf{t2w}$ and $\approx_{\mathsf{t2w}}$ that, if $((v, f), \mathbf{a}) \approx_{\mathsf{t2w}} (v, \mathbf{b})$, then for each variable $x_i \in \mathcal{X}$ such that $f(x_i) \neq \bot$, we have $\mathbf{a}_i = \mathbf{b}_i$. Observe that the control switches described above will mean that each variable $x_i \in \mathcal{X}$ is either reset to $post_i$, or retains the same value in $\mathbf{a}$ and $\mathbf{b}$. Thus it immediately follows that, for each variable $x_i \in \mathcal{X}$ such that $g(x_i) \neq \bot$, we have $\mathbf{a}'_i = \mathbf{b}'_i$. Furthermore, for all other variables $x_i \in \mathcal{X}$ such that $g(x_i) \in \mathcal{K}_W$, we must have $g(x_i) = \mathbf{b}_i$. Both of these facts are by the construction of T_W. Therefore, $((v', g), \mathbf{a}') \approx_{\mathsf{t2w}} (v', \mathbf{b}')$.

Let $C \in (S_W \cup S_{T_W})/_{\approx_{\mathsf{t2w}}}$ be an equivalence class of $\approx_{\mathsf{t2w}}$. Consider the distribution $\tilde{p}_v$ of the concurrent probabilistic system $\mathcal{S}_W$, derived from p^j_v, and the distribution $\tilde{p}_{(v,f)}$ of the concurrent probabilistic system $\mathcal{S}_{T_W}$, which is derived from $p^j_{(v,f)}$. Our aim is to show that $\tilde{p}_v[C] = \tilde{p}_{(v,f)}[C]$. Let $Tuples_{T_W}(C) \subseteq \mathsf{support}(p^j_{(v,f)})$ be the subset of tuples in the support of $p^j_{(v,f)}$ which, when applied to the state $((v, f), \mathbf{a})$, result in states in the $\approx_{\mathsf{t2w}}$ equivalence class C; that is,

$$Tuples_{T_W}(C) = \{((v', g), post, X) \in \mathsf{support}(p^j_{(v,f)}) \mid ((v', g), \mathbf{a}[X := post]) \in C\} .$$

Then, for each tuple $q \in Tuples_{T_W}(C)$, we can find a tuple $q' \in \mathsf{support}(p^j_v)$ which, by the argument given in the previous paragraph, results in a state in the equivalence class C when applied to the state $(v, \mathbf{b})$. Furthermore, by the construction of T_W, we have $p^j_{(v,f)}(q) = p^j_v(q')$. As $\approx_{\mathsf{t2w}}$ is an equivalence, we can also reverse the direction of the argument, and find the set $Tuples_W(C) \subseteq \mathsf{support}(p^j_v)$, which is defined analogously to $Tuples_{T_W}(C)$. Then

$$\tilde{p}_v[C] = \sum_{q \in Tuples_W(C)} p^j_v(q) = \sum_{q \in Tuples_{T_W}(C)} p^j_{(v,f)}(q) = \tilde{p}_{(v,f)}[C] .$$

That is, the same probability is assigned to states that are equivalent according to $\approx_{\mathsf{t2w}}$. We can repeat this process for all $1 \leq j \leq l$. Therefore, it follows from Definition 3.2.16 that $((v, f), \mathbf{a}) \simeq (v, \mathbf{b})$.

Our final task it to show that the initial states $\bar{s}_{T_W}$ and $\bar{s}_W$ of $\mathcal{S}_{T_W}$ and $\mathcal{S}_W$, respectively, are bisimilar. It follows from the definition of the initial conditions of T_W that $\bar{s}_{T_W} \approx_{\mathsf{t2w}} \bar{s}_W$, and therefore $\bar{s}_{T_W} \simeq \bar{s}_W$. □

We note that the equivalence $\approx_{\mathsf{t2w}}$ relates to a *timed* bisimulation between $\mathcal{S}_W$ and $\mathcal{S}_{T_W}$; that is, the bisimulation relation takes into account the exact duration of time transitions. This is not necessary for our PBTL model checking result; the equivalence $\approx_{\mathsf{t2w}}$ is also a *time-abstract* bisimulation (in which the time-abstract concurrent probabilistic systems $\mathcal{S}'_W$ and $\mathcal{S}'_{T_W}$ of W and T_W are used), which again preserves PBTL properties.

Verification of W against the PBTL property Φ is then performed by transforming W to T_W, and then model checking T_W against Φ using the method of Chapter 5. Then, if T_W satisfies (does not satisfy) the property Φ, by Theorem 3.3.6 we conclude that W satisfies (does not satisfy, respectively) Φ.

8.3 Translation: multisingular to stopwatch

This section presents a method for the transformation of an initialised probabilistic multisingular automaton into an equivalent initialised probabilistic stopwatch automaton, again with the notion of equivalence being probabilistic bisimulation. The transformation relies on appropriate scaling of the constants used in the constraints of the model, so that, for each variable $x_i \in \mathcal{X}$, after scaling, the associated flow condition is either $\dot{x}_i = 1$ or $\dot{x}_i = 0$. This technique is based on the results for non-probabilistic multisingular automata presented in [OSY94, ACH+95, HKPV98].

8.3.1 Definition of the translation: multisingular to stopwatch

Take a particular initialised probabilistic multisingular automaton

$$M = (\mathcal{X}, V, \mathcal{L}, init^M, inv^M, flow^M, prob^M, \langle pre_v^M \rangle_{v \in V}) \ .$$

Definition 8.3.1 (Scaling factor) *For the initialised probabilistic multisingular automaton M, and for all locations $v \in V$ and variables $x_i \in \mathcal{X}$ of M, if $flow^M(v)_i = [k,k]$ for some $k \in \mathbb{Z}$, then we define the* scaling factor of x_i in v *by k_i^v, where $k_i^v = k$ if $k \neq 0$, and $k_i^v = 1$ if $k = 0$.*

From M we construct the probabilistic stopwatch automaton

$$W_M = (\mathcal{X}, V, \mathcal{L}, init^{W_M}, inv^{W_M}, flow^{W_M}, prob^{W_M}, \langle pre_v^{W_M} \rangle_{v \in V})$$

in the following way. It will become apparent in the following description that W_M features rectangles with *rational* endpoints; however, the constants used in the description of W_M can be *scaled up* by a constant factor to obtain an initialised stopwatch automaton featuring rectangles with integer endpoints.

Initial condition. Recall that the initial condition of M is such that $init^M(v) \neq \emptyset$ for only one location $v \in V$, and $init^M(v) \in \mathbb{Z}^n$ (that is, $init^M(v)$ is a point in $\mathbb{Z}^n$-space). Then, for every variable $x_i \in \mathcal{X}$, if $init^M(v)_i = [k,k]$ for some $k \in \mathbb{Z}$, then we let $init^{W_M}(v)_i = [\frac{k}{k_i^v}, \frac{k}{k_i^v}]$.

Invariant conditions. Similarly, for each location $v \in V$ and each variable $x_i \in \mathcal{X}$, if $inv^M(v)_i = \langle\!\langle k_1, k_2 \rangle\!\rangle$, where $k_1, k_2 \in \mathbb{Z}$, $\langle\!\langle \in \{ \ (,[\ \}$, and $\rangle\!\rangle \in \{ \ ),] \ \}$, then $inv^{W_M}(v)_i = \langle\!\langle \frac{k_1}{k_i^v}, \frac{k_2}{k_i^v} \rangle\!\rangle$.

Flow conditions. Furthermore, for all locations $v \in V$ and all variables $x_i \in \mathcal{X}$, if $\mathit{flow}^M(v)_i = [0,0]$, then $\mathit{flow}^{W_M}(v)_i = [0,0]$, otherwise $\mathit{flow}^{W_M}(v)_i = [1,1]$.

Probability distributions. The distributions used in the description of W_M are similar to those used in M, except that the post-conditions of discrete transitions must be re-scaled. Recall that, for each location $v \in V$, we have $\mathit{prob}^M(v) = \{p_M^1, ..., p_M^l\}$ for some finite $l \in \mathbb{N}$. Then we let $\mathit{prob}^{W_M}(v) = \{p_{W_M}^1, ..., p_{W_M}^l\}$, where, for all $1 \leq j \leq l$, the distribution $p_{W_M}^j$ is derived from p_M^j in the following manner. Take the tuple $(v', \mathit{post}_M, X) \in V_M \times \mathbb{Z}^n \times 2^{\mathcal{X}}$, which is such that $p_M^j(v', \mathit{post}_M, X) > 0$. Then we derive the post-condition post_{W_M} from post_M by letting $\mathit{post}_{W_M} = \frac{k}{k_i^{v'}}$ if $(\mathit{post}_M)_i = k$, for each $x_i \in \mathcal{X}$. Next, we let $p_{W_M}^j(v', \mathit{post}_{W_M}, X) = p_M^j(v', \mathit{post}_M, X)$. This procedure is repeated for all tuples $(v', \mathit{post}_{W_M}, X)$ derived from those tuples (v', post_M, X) such that $p_M^j(v', \mathit{post}_M, X) > 0$, and we let $p_{W_M}^j$ assign probability 0 to all other tuples.

Pre-conditions. For all locations $v \in V$, where $\mathit{prob}^M(v) = \{p_M^1, ..., p_M^l\}$, and for all $1 \leq j \leq l$ and all variables $x_i \in \mathcal{X}$, we obtain $\mathit{pre}_v^{W_M}(p_{W_M}^j)_i$ from $\mathit{pre}_v^M(p_M^j)_i$ in the following way. If $\mathit{pre}^M(p_M^j)_i = \langle\langle k_1, k_2 \rangle\rangle$, where $k_1, k_2 \in \mathbb{Z}$, $\langle\langle \in \{ (, [\}$, and $\rangle\rangle \in \{),] \}$, then $\mathit{pre}_v^{W_M}(p_{W_M}^j)_i = \langle\langle \frac{k_1}{k_i^v}, \frac{k_2}{k_i^v} \rangle\rangle$.

8.3.2 Equivalence: multisingular and stopwatch

The concurrent probabilistic systems $\mathcal{S}_M = (S_M, \bar{s}_M, L_M, \Sigma_M, \mathit{Steps}_M)$ and $\mathcal{S}_{W_M} = (S_{W_M}, \bar{s}_{W_M}, L_{W_M}, \Sigma_{W_M}, \mathit{Steps}_{W_M})$ of M and W_M can be obtained using Definition 7.3.1. Furthermore, the function $\mathsf{w2m}$ of [HKPV98], which relates states of W_M and M, induces the equivalence relation $\approx_{\mathsf{w2m}}$ that is a bisimulation on $\mathcal{S}_M$ and $\mathcal{S}_{W_M}$, in the same way that $\mathsf{t2w}$ induced the bisimulation $\approx_{\mathsf{t2w}}$ in Section 8.2.2.

Definition 8.3.2 (cf. [HKPV98]) *Let M be an initialised probabilistic multisingular automaton, let W_M be an initialised probabilistic stopwatch automaton derived from M as in Section 8.3, let $\mathcal{S}_M$ and $\mathcal{S}_{W_M}$ be their concurrent probabilistic systems, and let $\{k_i^v \in \mathbb{Z} \mid v \in V_M, x_i \in \mathcal{X}\}$ be the set of scaling factors of M. Then let $\mathsf{w2m} : S_{W_M} \to S_M$ be the bijection such that, for each $(v, \mathbf{a}) \in S_{W_M}$, we have $\mathsf{w2m}(s) = (v, \mathbf{b})$, where $\mathbf{b}_i = k_i^v.\mathbf{a}_i$ for each variable $x_i \in \mathcal{X}$.*

The relation $\approx_{\mathsf{w2m}}$ is an equivalence on $S_{W_M} \cup S_M$ such that $s \approx_{\mathsf{w2m}} s'$ if and only if $\mathsf{w2m}(s) = s'$ for the states $s \in S_M$ and $s' \in S_{W_M}$.

Proposition 8.3.3 *Let M be an initialised probabilistic multisingular automaton, and let W_M be the initialised probabilistic stopwatch automaton derived from M using the translation method of Section 8.3. Then the equivalence $\approx_{\mathsf{w2m}}$ is a bisimulation between $\mathcal{S}_M$ and $\mathcal{S}_{W_M}$.*

Proof. To prove the proposition, we first show that, for any states $(v, \mathbf{a}) \in S_{W_M}$ and $(v, \mathbf{b}) \in S_M$ are such that $(v, \mathbf{a}) \approx_{\mathsf{t2w}} (v, \mathbf{b})$, then $(v, \mathbf{a}) \simeq (v, \mathbf{b})$. We deal with the conditions of the definition of bisimulation (Definition 3.2.16) in turn.

Firstly, the requirement of bisimulation that $L_{W_M}(v, \mathbf{a}) = L_W(v, \mathbf{b})$ can be shown in the same manner as in Proposition 8.2.2.

Secondly, we now show that there exists the transition $(v, \mathbf{a}) \xrightarrow{\sigma, \tilde{p}}$ if and only if there exists the transition $(v, \mathbf{b}) \xrightarrow{\sigma, \tilde{p}'}$, where $\tilde{p} \approx_{\mathsf{w2m}} \tilde{p}'$. As in Proposition 8.2.2, there are two cases, depending on whether a time transition or a discrete transition is made.

Time transition. We first consider the case of time transitions; that is, we show that there exists the transition $(v, \mathbf{a}) \xrightarrow{\delta} (v, \mathbf{c})$ of $\mathcal{S}_{W_M}$ if and only if there exists the transition $(v, \mathbf{b}) \xrightarrow{\delta} (v, \mathbf{d})$ of $\mathcal{S}_M$. Again, as in Proposition 8.2.2, we proceed by showing that $(v, \mathbf{c}), (v, \mathbf{d})$, and all intermediate states visited during the time transition, are admissible states, and that $(v, \mathbf{c}) \approx_{\mathsf{w2m}} (v, \mathbf{d})$. Consider the variable $x_i \in \mathcal{X}$. As $(v, \mathbf{a}) \approx_{\mathsf{w2m}} (v, \mathbf{b})$, we have $\mathbf{b}_i = k_i^v.\mathbf{a}_i$. We deal with two cases depending on the flow condition $\mathit{flow}^M(v)_i$ of the location v of M.

Case: $\mathit{flow}^M(v)_i = [k, k]$**, where** $k \neq 0$. By the definition of the time transitions of $\mathcal{S}_{W_M}$, we have $\mathbf{c}_i = \mathbf{a}_i + \delta$. Similarly, by the definition of the time transitions of $\mathcal{S}_M$, and the definition of the scaling factor k_i^v, we have

$$\mathbf{d}_i = \mathbf{b}_i + k_i^v.\delta = k_i^v.\mathbf{a}_i + k_i^v.\delta = k_i^v(\mathbf{a}_i + \delta) \ .$$

Therefore, $\mathbf{d}_i = k_i^v.\mathbf{c}_i$.

Consider the case in which the interval $\mathit{inv}^M(v)_i = [k_1, k_2]$ for some $k_1, k_2 \in \mathbb{Z}$ (the cases in which $\mathit{inv}^M(v)_i$ is left-open and right-closed, right-open and left-closed, and open, follow naturally). Then, by definition of $\mathit{inv}^{W_M}(v)_i$, we have $\mathit{inv}^{W_M}(v)_i = [\frac{k_1}{k_i^v}, \frac{k_2}{k_i^v}]$. It is clear that $k_1 \leq k_i^v.\mathbf{c}_i \leq k_2$ if and only if $\frac{k_1}{k_i^v} \leq \mathbf{d}_i \leq \frac{k_2}{k_i^v}$. As in Proposition 8.2.2, by the convexity of the sets $\mathit{inv}^{W_M}(v)_i$ and $\mathit{inv}^M(v)_i$, we have $\mathbf{a}_i + \delta' \in \mathit{inv}^{W_M}(v)_i$ and $\mathbf{b}_i + k_i^v.\delta' \in \mathit{inv}^M(v)_i$ for all $0 \leq \delta' \leq \delta$.

Case: $\mathit{flow}^M(v)_i = [0, 0]$. By the definition of time transitions of $\mathcal{S}_{W_M}$ and $\mathcal{S}_M$, we must have $\mathbf{d}_i = \mathbf{b}_i$ and $\mathbf{c}_i = \mathbf{a}_i$. As $\mathbf{b}_i = k_i^v.\mathbf{a}_i$, we can write $\mathbf{d}_i = k_i^v.\mathbf{c}_i$.

Then, as $\mathbf{d}_i = k_i^v.\mathbf{c}_i$ holds for all variables $x_i \in \mathcal{X}$, we can write $(v, \mathbf{c}) \approx_{\mathsf{w2m}} (v, \mathbf{d})$, proving the required result for time transitions.

Discrete transition. Next, we show that there exists a discrete transition of $\mathcal{S}_{W_M}$ denoted by $(v, \mathbf{a}) \xrightarrow{\theta, \tilde{p}}$ if and only if there exists a discrete transition of $\mathcal{S}_M$ denoted by $(v, \mathbf{b}) \xrightarrow{\theta, \tilde{p}'}$ such that $\tilde{p} \approx_{\mathsf{w2m}} \tilde{p}'$. First we show that, for some $1 \leq j \leq |\mathit{prob}^M(v)|$, if $p_M^j \in \mathit{prob}^M(v)$ is enabled in $(v, \mathbf{b})$, then the distribution $p_{W_M}^j \in \mathit{prob}^{W_M}(v)$, where $p_{W_M}^j$ is derived from p_M^j in the manner of the construction of W_M, is also enabled. For simplicity, we deal with the case in which the interval $\mathit{pre}_v^M(p_M^j) = [k_1, k_2]$ is closed. By the construction of W_M, we have $\mathit{pre}_v^{W_M}(p_{W_M}^j) = [\frac{k_1}{k_i^v}, \frac{k_2}{k_i^v}]$. Then, as $\mathbf{b}_i = k_i^v.\mathbf{a}_i$, it follows that the statements $k_1 \leq \mathbf{b}_i \leq k_2$ and $\frac{k_1}{k_i^v} \leq \mathbf{a}_i \leq \frac{k_2}{k_i^v}$ are equivalent.

Our next task is to show that, for every tuple $(v', post_M, X) \in \mathsf{support}(p^j_M)$, and the tuple $(v', post_{W_M}, X) \in \mathsf{support}(p^j_{W_M})$ which it derives, are such that we have $(v', \mathbf{a}[X := post_M]) \approx_{\mathsf{w2m}} (v', \mathbf{b}[X := post_{W_M}])$. As the locations of these two states are identical, and by the definition of $\approx_{\mathsf{w2m}}$, our problem reduces to showing that $k^{v'}_i.\mathbf{a}[X := post_{W_M}]_i = \mathbf{b}[X := post_M]_i$ for each variable $x_i \in \mathcal{X}$. Let $(post_M)_i = k$ for $k \in \mathbb{Z}$. From the construction of W_M, if $x_i \in X$, then $k^{v'}_i.\mathbf{a}[X := post_{W_M}]_i = k^{v'}_i \cdot \frac{k}{k^{v'}_i} = k$, and $\mathbf{b}[X := post_M]_i = k$. Therefore, $k^{v'}_i.\mathbf{a}[X := post_{W_M}]_i = \mathbf{b}[X := post_M]_i$. On the other hand, if $x_i \notin X$, then, because M is initialised, $\mathit{flow}^M(v')_i = \mathit{flow}^M(v)_i$, and therefore $k^{v'}_i = k^v_i$. From this fact and from the construction of W_M, we have $k^{v'}_i.\mathbf{a}[X := post_{W_M}]_i = k^v_i.\mathbf{a}_i$, and $\mathbf{b}[X := post_M]_i = \mathbf{b}_i$. Then, from $k^v_i.\mathbf{a}_i = \mathbf{b}_i$, we have $k^{v'}_i.\mathbf{a}[X := post_{W_M}]_i = \mathbf{b}[X := post_M]_i$.

Let $\tilde{p}_M$ be the distribution of the concurrent probabilistic system $\mathcal{S}_M$ which is derived from p_M, and let $\tilde{p}_{W_M}$ be the distribution of the concurrent probabilistic system $\mathcal{S}_{W_M}$ derived from p_{W_M}. This correspondence allows us to show that, for every $\approx_{\mathsf{w2m}}$ equivalence class $C \in (S_W \cup S_{W_M})/_{\approx_{\mathsf{w2m}}}$, we have the equality $\tilde{p}_M[C] = \tilde{p}_{W_M}[C]$, using a similar argument to that employed in the proof of Proposition 8.2.2.

Finally, we show that the initial states $\bar{s}_{W_M}$ and $\bar{s}_M$ of $\mathcal{S}_{W_M}$ and $\mathcal{S}_M$, respectively, are bisimilar. From the definition of the initial condition of W_T, we have the fact that $\bar{s}_{W_M} \approx_{\mathsf{w2m}} \bar{s}_M$, and therefore $\bar{s}_{W_M} \simeq \bar{s}_M$. Hence, $\mathcal{S}_{W_M}$ and $\mathcal{S}_M$ are bisimilar concurrent probabilistic systems. □

The model checking process for an initialised probabilistic multisingular automaton M then comprises the following steps:

1. transforming M to the initialised probabilistic stopwatch automaton W_M,
2. scaling up the rational constants used in the description of W_M to integers in order to obtain the additional initialised probabilistic stopwatch automaton W'_M,
3. transforming W'_M to the probabilistic timed automaton denoted by $T_{W'_M}$.
4. finally, model checking $T_{W'_M}$ against the PBTL formula Φ. Then, by Theorem 3.3.6, the probabilistic timed automaton $T_{W'_M}$ satisfies ϕ if and only if the probabilistic multisingular automaton M satisfies ϕ.

8.4 Translation: rectangular to multisingular

We now introduce a model checking strategy for verifying initialised probabilistic rectangular automata against $\forall$PBTL properties based on similar results in the non-probabilistic context of [OSY94, HKPV98]. The approach taken is to construct, from an initialised probabilistic rectangular automaton R, an initialised probabilistic multisingular automaton M_R, such that model checking of the latter allows us to conclude

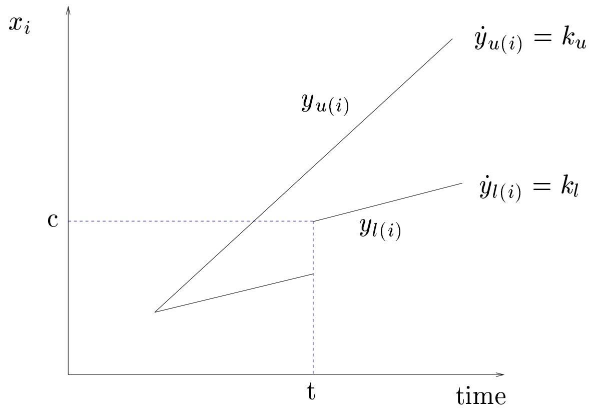


Figure 8.1: Updating the value of x_i after a control switch at time t.

results about the former. More precisely, each variable $x_i \in \mathcal{X}$ of R is represented by two variables $y_{l(i)}, y_{u(i)} \in \mathcal{Y}$ of M_R, with the intuition that $y_{l(i)}$ tracks the least possible value of x_i, whereas $y_{u(i)}$ tracks its greatest possible value. Therefore, the singular flow conditions for $y_{l(i)}$ are derived from the minimal slopes that x_i may take in R, whereas the flow conditions for $y_{u(i)}$ are derived from x_i's maximal slopes. Furthermore, the probabilistic edge relation of M_R is defined to update $y_{l(i)}$ and $y_{u(i)}$ so that the interval $[y_{l(i)}, y_{u(i)}]$ correctly represents the possible values of x_i.

The use of the variables $y_{l(i)}$ and $y_{u(i)}$ can be understood with the help of Figure 8.1. Say the current location is v, and that the flow condition $\mathit{flow}(v)_i$ of the variable x_i is the interval $[k_l, k_u]$. As time passes, the possible values of x_i are contained within an envelope, the lower (respectively, upper) bound of which is represented by $y_{l(i)}$ (respectively, $y_{u(i)}$). If a control switch occurs, the values of $y_{l(i)}$ and $y_{u(i)}$ must continue to represent the possible values of x_i; this may involve resetting $y_{l(i)}$ or $y_{u(i)}$, or both, even if x_i is not reset by the corresponding control switch in R. In Figure 8.1, at time t a distribution p is chosen, where $\mathit{pre}_v(p) = [c, \infty)$, and say a tuple $(v', \mathit{post}, X) \in \mathsf{support}(p)$ is probabilistically selected for which $x_i \notin X$. Then $y_{l(i)}$ must be updated to the value c when emulating this control switch, as its value is below c when the distribution p was selected. This reflects the standard intuition in the non-probabilistic case of [HKPV98].

8.4.1 Definition of the translation: rectangular to multisingular

The construction of M_R is subject to a simplifying assumption, namely that, for all locations $v \in V$, $\mathit{inv}^R(v)$ and $\mathit{flow}^R(v)$ are compact rectangles, for all $p \in \mathit{prob}^R(v)$, the rectangle $\mathit{pre}_v^R(p)$ is compact, and for each $(v', \mathit{post}, X) \in \mathsf{support}(p)$, the rectangle post is compact. Note that it follows from [HKPV98] that *all* of these assumptions may be dropped; however, in such a case, the construction of M_R requires significant book-work that is independent of probabilistic concerns.

Let $R = (\mathcal{X}, V, L, \mathit{init}^R, \mathit{inv}^R, \mathit{flow}^R, \mathit{prob}^R, \langle \mathit{pre}^R_v \rangle_{v \in V})$ be an initialised probabilistic rectangular automaton subject to the assumptions made above. We construct the probabilistic multisingular automaton $M_R = (\mathcal{Y}, V, L, \mathit{init}^{M_R}, \mathit{inv}^{M_R}, \mathit{flow}^{M_R}, \mathit{prob}^{M_R}, \langle \mathit{pre}^{M_R}_v \rangle_{v \in V})$ that represents R in the following way.

Variables Let $\mathcal{Y} = \{y_1, ..., y_{2n}\}$, where the $l(i)$-th variable $y_{l(i)}$ represents the lower bound on the i-th variable $x_i \in \mathcal{X}$ of R, the $u(i)$-th variable $y_{u(i)}$ represents the upper bound on $x_i \in \mathcal{X}$, and $l(i) = 2i - 1$, $u(i) = 2i$.

Initial and invariant sets. For each location $v \in V$, let $\mathit{init}^{M_R}(v)_{l(i)} = \mathit{init}^{M_R}(v)_{u(i)} = \mathit{init}^R(v)_i$. Furthermore, for each variable $x_i \in \mathcal{X}$, if $\mathit{inv}^R(v)_i = [l, u]$, then $\mathit{inv}^{M_R}_{l(i)} = [l, \infty)$ and $\mathit{inv}^{M_R}_{u(i)} = (-\infty, u]$.

Flow inclusion. For each location $v \in V$ and $x_i \in \mathcal{X}$, if $\mathit{flow}^R(v)_i = [l, u]$ then $\mathit{flow}^{M_R}(v)_{l(i)} = [l, l]$ and $\mathit{flow}^{M_R}(v)_{u(i)} = [u, u]$.

The components of M_R described above follow the definitions given in the non-probabilistic context of [OSY94, HKPV98]. However, the subsequent definition of the functions prob^{M_R} and $\langle \mathit{pre}^{M_R}_v \rangle_{v \in V}$ are substantially more involved, and require preliminary explanation. Again, refer to Figure 8.1, and assume that x_i is the only variable of the (probabilistic) rectangular automaton R. Then the (probabilistic) multisingular automaton M_R copies the discrete transition at time t by choosing one of its distributions. This distribution must encode the fact that, as the value of $y_{l(i)}$ was below c when the distribution was taken, the variable $y_{l(i)}$ must be updated to c. However, now consider the situation in which the value of $y_{l(i)}$ was *above* c at time t; this may have been the case if this location was entered with a higher value of $y_{l(i)}$. Then, the value of $y_{l(i)}$ should *not* be reset to c when the transition is made. This fact can be encoded into a second distribution of M_R, which is enabled when $y_{l(i)}$ is greater than c, and which does not involve the value of $y_{l(i)}$ being reset to c.

Similarly, if the pre-condition of the transition of R also involved an *upper bound*, then we would have to consider similar reasoning, now applied to the value of $y_{u(i)}$. That is, we would consider two distributions of M_R to represent the transition; one which resets the value of $y_{u(i)}$ to the relevant upper bound, and another which does not. Note that we then have four possibilities: two such possibilities correspond to the case in which the value of $y_{l(i)}$ is below the pre-condition's lower bound, with the first corresponding to $y_{u(i)}$ being above the pre-condition's upper bound, the second to the case in which it is not. Similarly, we have two possibilities for the case in which the value of $y_{l(i)}$ is above the lower bound, one for each of the possibilities concerning $y_{u(i)}$. Therefore, we represent a single distribution of R with *four* distributions of M_R.

However, when moving to the case in which the (probabilistic) rectangular automaton R has *two* variables, we must represent a single distribution of R by *sixteen* distributions of M_R. That is, we have a distribution for each of the four types of distributions described above for one variable, combined with four types of distributions for the other variable. Therefore, with $n \in \mathbb{N}$ variables of R, we should consider representing every distribution of R with 4^n distributions of M_R. To uniquely identify each distribution, we associate it with a *status*, which takes the form of a function assigning a number between 1 and 4 to each variable of R. Then, for example, the number 1 assigned to the variable $x_i \in \mathcal{X}$ corresponds to the lower bound of x_i being below the lower bound of the relevant pre-condition, and the upper bound of x_i being below the upper bound of the pre-condition.

Probability distributions. For each location $v \in V$ and $p^R \in \mathit{prob}^R(v)$, there exists a corresponding set of distributions that are derived from p^R, denoted by $\mathsf{derive}(p^R) \subseteq \mathit{prob}^{M_R}(v)$, the elements of which take the following form. Let $\mathsf{status} : \mathcal{X} \to \{1, ..., 4\}$ be a function assigning a *status* to each variable $x_i \in \mathcal{X}$, where the set of all such functions is denoted by $4^{\mathcal{X}}$. Then there exists a distribution $p^{M_R}_{\mathsf{status}} \in \mathsf{derive}(p^R)$ for each status function $\mathsf{status} \in 4^{\mathcal{X}}$.

We turn our attention to the tuples in the support of each distribution $p^{M_R}_{\mathsf{status}} \in \mathsf{derive}(p^R)$. For each tuple $(v', \mathit{post}^R, X) \in \mathsf{support}(p^R)$, there exists a corresponding tuple $(v', \mathit{post}^{M_R}_{\mathsf{status}}, Y_{\mathsf{status}}) \in \mathsf{support}(p^{M_R}_{\mathsf{status}})$. Let $[l_i, u_i] = \mathit{pre}_v(p^R)_i$ and $[l'_i, u'_i] = (\mathit{post}^R)_i$ for each $1 \leq i \leq n$. Consider the following two cases.

Case: $x_i \in X$**.** We let $(\mathit{post}^{M_R}_{\mathsf{status}})_{l(i)} = l'_i$, $(\mathit{post}^{M_R}_{\mathsf{status}})_{u(i)} = u'_i$, and $y_{l(i)}, y_{u(i)} \in Y_{\mathsf{status}}$ for each $\mathsf{status} \in 4^{\mathcal{X}}$.

Case: $x_i \notin X$**.** We have four sub-cases depending on the value of $\mathsf{status}(x_i)$. We use *arb* to denote an arbitrary (natural number) value.

Sub-case: $\mathsf{status}(x_i) = 1$**.** Let:

$$(\mathit{post}^{M_R}_{\mathsf{status}})_{l(i)} = l_i, \quad (\mathit{post}^{M_R}_{\mathsf{status}})_{u(i)} = \mathit{arb}, \quad y_{l(i)} \in Y_{\mathsf{status}} \ .$$

Sub-case: $\mathsf{status}(x_i) = 2$**.** Let:

$$(\mathit{post}^{M_R}_{\mathsf{status}})_{l(i)} = l_i, \quad (\mathit{post}^{M_R}_{\mathsf{status}})_{u(i)} = u_i, \quad y_{l(i)}, y_{u(i)} \in Y_{\mathsf{status}} \ .$$

Sub-case: $\mathsf{status}(x_i) = 3$**.** Let:

$$(\mathit{post}^{M_R}_{\mathsf{status}})_{l(i)} = \mathit{arb}, \quad (\mathit{post}^{M_R}_{\mathsf{status}})_{u(i)} = \mathit{arb}; \text{no requirement on } Y_{\mathsf{status}}.$$

Sub-case: $\mathsf{status}(x_i) = 4$**.** Let:

$$(\mathit{post}^{M_R}_{\mathsf{status}})_{l(i)} = \mathit{arb}, \quad (\mathit{post}^{M_R}_{\mathsf{status}})_{u(i)} = u_i, \quad y_{u(i)} \in Y_{\mathsf{status}} \ .$$

We say that the tuple $(v', post^{M_R}_{\mathsf{status}}, Y_{\mathsf{status}})$ is *obtained* from the tuple $(v', post^R, X)$, written $(v', post^R, X) \rightsquigarrow (v', post^{M_R}_{\mathsf{status}}, Y_{\mathsf{status}})$.

Finally, given that we have obtained a set of tuples from all of the tuples in $\mathsf{support}(p^R)$, we can define the probabilities which $p^{M_R}_{\mathsf{status}}$ assigns to the former set. Note that it is possible that a single tuple in $\mathsf{support}(p^{M_R}_{\mathsf{status}})$ is obtained from more than one tuple in $\mathsf{support}(p^R)$. Therefore, we compute the probability of the choice of a tuple in $\mathsf{support}(p^{M_R}_{\mathsf{status}})$ according to the following sum:

$$p^{M_R}_{\mathsf{status}}(v', post^{M_R}_{\mathsf{status}}, Y_{\mathsf{status}}) = \sum_{\substack{(v', post^R, X) \in \mathsf{support}(p^R) \\ \& \ (v', post^R, X) \rightsquigarrow (v', post^{M_R}_{\mathsf{status}}, Y_{\mathsf{status}})}} p^R(v', post^R, X) \ .$$

We then repeat this process for all $p^{M_R}_{\mathsf{status}} \in \mathsf{derive}(p^R)$.

Pre-condition sets. For every location $v \in V$ and distribution $p^R \in prob^R(v)$, we define the pre-condition sets for $\mathsf{derive}(p^R) \subseteq prob^{M_R}(v)$ in the following way. For every $x_i \in \mathcal{X}$, if $pre^R_v(p^R)_i = [l, u]$, then for each $p^{M_R}_{\mathsf{status}} \in \mathsf{derive}(p^R)$:

if $\mathsf{status}(x_i) = 1$, then let
$$pre^{M_R}_v(p^{M_R}_{\mathsf{status}})_{l(i)} = (-\infty, l), \quad pre^{M_R}_v(p^{M_R}_{\mathsf{status}})_{u(i)} = [l, u];$$
if $\mathsf{status}(x_i) = 2$, then let
$$pre^{M_R}_v(p^{M_R}_{\mathsf{status}})_{l(i)} = (-\infty, l), \quad pre^{M_R}_v(p^{M_R}_{\mathsf{status}})_{u(i)} = (u, \infty);$$
if $\mathsf{status}(x_i) = 3$, then let
$$pre^{M_R}_v(p^{M_R}_{\mathsf{status}})_{l(i)} = [l, u], \quad pre^{M_R}_v(p^{M_R}_{\mathsf{status}})_{u(i)} = [l, u];$$
if $\mathsf{status}(x_i) = 4$, then let
$$pre^{M_R}_v(p^{M_R}_{\mathsf{status}})_{l(i)} = [l, u], \quad pre^{M_R}_v(p^{M_R}_{\mathsf{status}})_{u(i)} = (u, \infty).$$

This completes the definition of the method for constructing the probabilistic multisingular automaton M_R from the probabilistic rectangular automaton R. From M_R, we can now derive its associated concurrent probabilistic system $\mathcal{S}_{M_R}$. However, for purely technical reasons, we take an unconventional approach when labelling discrete transitions with events; instead of labelling such transitions with the distribution p^{M_R} of M_R from which they are obtained, we label discrete transitions *with the distribution of R which is used to derive p^{M_R}*. This allows us to define an appropriate simulation relation between R and M_R in the next section.

Definition 8.4.1 (Concurrent probabilistic system of M_R) *Let R be a probabilistic rectangular automaton, and M_R be the probabilistic multisingular automaton constructed from R using the above method. Then we define the concurrent probabilistic system $\mathcal{S}_{M_R} = (S_{M_R}, \bar{s}_{M_R}, L_{M_R}, \Sigma_{M_R}, Steps_{M_R})$ of M_R, where S_{M_R}, $\bar{s}_{M_R}$, and L_{M_R} are defined according to Definition 7.3.1. Let $\Sigma_{M_R} = \Sigma_R$, where Σ_R is the event set of the concurrent probabilistic system $\mathcal{S}_R$ of R. Furthermore, the function $Steps_{M_R}$ is*

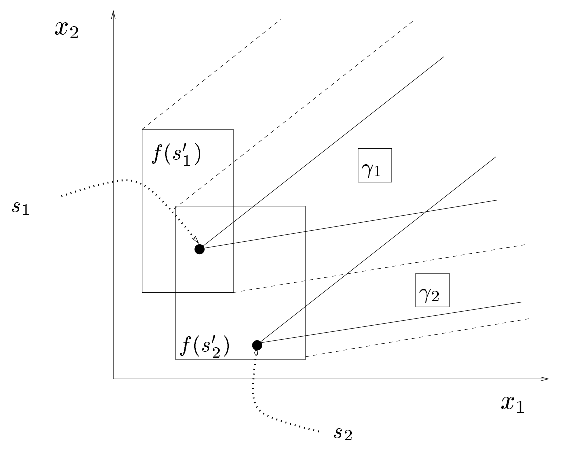


Figure 8.2: M_R simulates R.

defined as in Definition 7.3.1, except that, for every state $s \in \mathcal{S}_{M_R}$, we replace every event-distribution pair of the form $(\theta_{p^{M_R}}, \tilde{p}) \in Steps_{M_R}(s)$ with an event-distribution pair $(\theta_{p^R}, \tilde{p})$, where p^R is the unique distribution of R such that $p^{M_R} \in \mathsf{derive}(p^R)$.

The function f of [HKPV98], which relates a set of states of R to a state of M_R, can also be used in the probabilistic context.

Definition 8.4.2 (cf. [HKPV98]) *Let R be a probabilistic rectangular automaton, M_R be the probabilistic multisingular automaton constructed from R using the above method, and $\mathcal{S}_R$ and $\mathcal{S}_{M_R}$ be their associated concurrent probabilistic systems. Then the function $f : S_{M_R} \to 2^{S_R}$ is such that, for any state $(v, \mathbf{a}) \in S_R$ of $\mathcal{S}_R$, we have $f(v, \mathbf{a}) = \{v\} \times (\Pi_{i=1}^{n}[\mathbf{a}_{l(i)}, \mathbf{a}_{u(i)}] \cap inv^R(v))$.*

Then the relation $\mathcal{R}_f \subseteq S_R \times S_{M_R}$ is defined in the following manner: for any $s \in S_R$ and $s' \in S_{M_R}$ such that $s \in f(s')$, then $s\mathcal{R}_f s'$.

8.4.2 The relationship between the rectangular and multisingular systems

The strategy for model checking R via the construction M_R, as suggested by the existing literature for non-probabilistic hybrid automata [OSY94, HKPV98], is based on the interpretation of M_R as an *abstraction* of R. More precisely, the concurrent probabilistic system $\mathcal{S}_R$ is simulated by the concurrent probabilistic system $\mathcal{S}_{M_R}$; then, by Theorem 3.3.5, if a $\forall$PBTL formula Φ is satisfied by $\mathcal{S}_{M_R}$, this is a sufficient condition for concluding that Φ is also satisfied by $\mathcal{S}_R$. This simulation relation applies both to the timed and time-abstract concurrent probabilistic systems of $\mathcal{S}_R$ and $\mathcal{S}_{M_R}$. In this section, we establish that the relation $\mathcal{R}_f$ is a simulation between $\mathcal{S}_R$ and $\mathcal{S}_{M_R}$.

An example of the way in which M_R simulates the R is shown in Figure 8.2. The variable set of R contains two variables, x_1 and x_2. Consider a state $s \in S_R$ of R, and a state $s' \in S_{M_R}$ of M_R which are such that $s \in f(s')$, and therefore $s\mathcal{R}_f s'$. Say R nondeterministically selects an enabled distribution p^R for choice, which, by construction of M_R, can be matched in s' by one of the distributions in the set $\mathsf{derive}(p^R)$. As the pre-conditions of each distribution in $\mathsf{derive}(p^R)$ are disjoint, we choose arbitrarily one of the distributions $p^{M_R}_{\mathsf{status}}$, for some $\mathsf{status} \in 4^{\mathcal{X}}$. Henceforth, we omit the status subscript from $p^{M_R}_{\mathsf{status}}$ for simplicity. Let $p^R(v', post^R_1, X_1) = \frac{1}{3}$, $p^R(v', post^R_2, X_2) = \frac{1}{6}$, and $p^R(v', post^R_3, X_3) = \frac{1}{2}$. From the construction of M_R, we have a tuple $(v', post^{M_R}_i, Y_i)$ in the support of p^{M_R} for each tuple $(v', post^R_i, X_i)$ in the support of p^R, for $1 \leq i \leq 3$. Furthermore, tuples with the same subscript correspond to the same probability. That is, $p^{M_R}(v', post^{M_R}_1, Y_1) = \frac{1}{3}$, $p^{M_R}(v', post^{M_R}_2, Y_2) = \frac{1}{6}$, $p^{M_R}(v', post^{M_R}_3, Y_3) = \frac{1}{2}$ for $post^{M_R}_1, Y_1, post^{M_R}_2, Y_2, post^{M_R}_3, Y_3$ defined as in the construction process for M_R.

Now consider the possibility that $post^R_2, X_2$ and $post^R_3, X_3$ are such that, when applied to s, they result in the same target *sets* of states. Then the maximal and minimal values of x_1, x_2 encoded in M_R will be the same for $post^R_2, X_2$ and $post^R_3, X_3$, and therefore the probability of M_R making a transition to the state s'_2, which encodes these values, will be $\tilde{p}^{M_R}(s'_2) = p^{M_R}(v', post^{M_R}_2, Y_2) + p^{M_R}(w, post^{M_R}_3, Y_3) = \frac{1}{6} + \frac{1}{2} = \frac{2}{3}$. We encode the maximal and minimal values reached by M_R from s' after the probabilistic choice of $(v', post^{M_R}_1, Y_1)$ by the state of M_R denoted by s'_1; therefore, $\tilde{p}^{M_R}(s'_1) = p^{M_R}(v', post^{M_R}_1, Y_1) = \frac{1}{3}$. Say the rectangular sets encoded by s'_1 and s'_2, denoted by $f(s'_1)$ and $f(s'_2)$ respectively, overlap (see Figure 8.2).

Now we turn our attention to the transition made by R. Consider the case in which, after probabilistically choosing either of the tuples $(v', post^R_1, X_1)$ and $(v', post^R_2, X_2)$, the *same* target state s_1 is selected by R, which, naturally, is in the intersection of the state sets defined by $f(s'_1)$ and $f(s'_2)$. Next, we let the choice of target state after the probabilistic choice of $(v', post^R_3, X_3)$ be s_2, which is in $f(s'_2)$ but not $f(s'_1)$. Then, from our view of f as inducing the relation $\mathcal{R}_f$, we have $s_1 \mathcal{R}_f s'_1$ and $s_1, s_2 \mathcal{R}_f s'_2$. Say $\tilde{p}^R$ is the distribution of $\mathcal{S}_R$ corresponding to the choices of the target states s_1 and s_2; then $\tilde{p}^R(s_1) = p^R(v', post^R_1, X_1) + p^R(v', post^R_2, X_2) = \frac{1}{3} + \frac{1}{6} = \frac{1}{2}$, and $\tilde{p}^R(s_2) = p^R(v', post^R_3, X_3) = \frac{1}{2}$. To see that $\tilde{p}^R \mathcal{R}_f \tilde{p}^{M_R}$, the following weight function w, which relates $\tilde{p}^R$ and $\tilde{p}^{M_R}$ via $\mathcal{R}_f$, is defined: let $w(s_1, s'_1) = \frac{1}{3}, w(s_1, s'_2) = \frac{1}{6}, w(s_2, s'_2) = \frac{1}{2}$. The distributions $\tilde{p}^R$ and $\tilde{p}^{M_R}$, and the probabilities that they assign to the states in their support, are shown in Figure 8.3. The states in the relation $\mathcal{R}_f$ are linked by dotted lines. Finally, the weight function w is also represented pictorially in Figure 8.3.

The argument that f induces a simulation with regard to *continuous* transitions follows from the non-probabilistic precedent of [OSY94, HKPV98]. In Figure 8.2, both s'_1 and s'_2 can simulate all of the transitions from any state lying in the intersection $f(s'_1) \cap f(s'_2)$, such as s_1. For example, the distribution with the pre-condition rep-

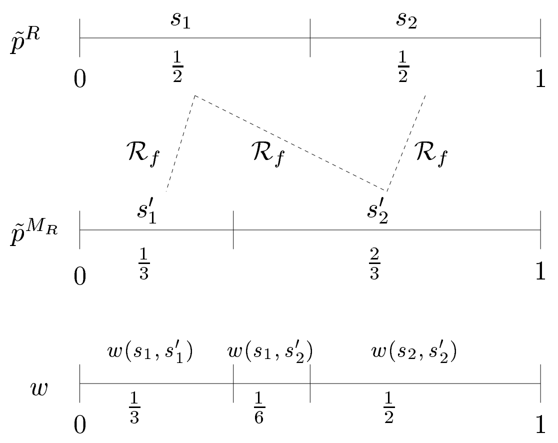


Figure 8.3: The distributions $\tilde{p}^R$ and $\tilde{p}^{M_R}$, and the weight function w.

resented by the rectangle γ_1 is enabled in s_1, s'_1 and s'_2, after some time has elapsed. However, as s_2 is in the state set $f(s'_2)$ but not $f(s'_1)$, the state s'_2, but not s'_1, can simulate the choice of the distribution with the pre-condition γ_2 by s_2.

We now formalise these notions in the following lemma. Note that we henceforth drop the subscript on M_R to simplify notation.

Lemma 8.4.3 *Let R be a probabilistic rectangular automaton, and M be the probabilistic multisingular automaton constructed from R. Let $\mathcal{S}_R$ and $\mathcal{S}_M$ be the concurrent probabilistic systems of R and M respectively. Let $s_1 \in S_M$ be a state of $\mathcal{S}_M$, and let $s_2 \in f(s_1)$ be a state of $\mathcal{S}_R$. Then $s_1 \preceq s_2$.*

The remainder of this section will be devoted to the proof of Lemma 8.4.3. We proceed by proving the following properties in turn.

A – Continuous transitions. For the states $s_1 \in S_R$, $s_2 \in S_M$ which are such that $s_1 \mathcal{R}_f s_2$, every continuous transition from s_1 can be matched by a continuous transition with the same duration from s_2, such that the target states of these transitions are also in the relation $\mathcal{R}_f$.

B – Discrete transitions. For the states $s_1 \in S_R$, $s_2 \in S_M$ which are such that $s_1 \mathcal{R}_f s_2$, every discrete transition from s_1 can be matched by a discrete transition with the same event label from s_2. To ease notation, we let r be the distribution of R, and let $\tilde{r}$ be the distribution of the underlying concurrent probabilistic system $\mathcal{S}_R$, that are used to make the transition from s_1. Similarly, let m be the distribution of M, and $\tilde{m}$ be the distribution of the concurrent probabilistic system $\mathcal{S}_M$, that are used to make the transition from s_2. We prove two related properties concerning these discrete transitions.

1. The target state of the transition of s_1 will be related via $\mathcal{R}_f$ to the target state of the transition of s_2, if the transitions are derived from "related" tuples in the support of r and m. It transpires that, to copy the transition from s_1 of R made according to the distribution r, we use a transition of M from the set $\mathsf{derive}(r)$. Recall that, by the definition of M, as the distribution m of M is derived from the distribution r of R, every tuple in the support of m corresponds to at least one tuple in the support of r. Then a tuple in the support of m is said to be "related" to a tuple in the support of r if the former is obtained from the latter.
2. The distributions $\tilde{r}$ of $\mathcal{S}_R$ and $\tilde{m}$ of $\mathcal{S}_M$ are such that $\tilde{r}\mathcal{R}_f\tilde{m}$. That is, there exists a weight function for $(\tilde{r}, \tilde{m})$ with respect to $\mathcal{R}_f$. The proof of this property requires a study of the tuples of the distributions r and m, and the way in which they interrelate with the states in the support of $\tilde{r}$ and $\tilde{m}$.

C – Simulation. The states $s_1 \in S_R$, $s_2 \in S_M$ which are such that $s_1\mathcal{R}_f s_2$ are also such that $s_1 \preceq s_2$.

The first two properties will be expressed as two sub-lemmas, and illustrated with examples. We should note that the machinery used to prove the property concerning continuous transitions, and the first property of discrete transitions, is based on the non-probabilistic precedent of [HKPV98].

A – Continuous transitions

We commence with a lemma stating that the required property concerning continuous transitions.

Lemma 8.4.4 (Continuous transitions) *Let R be a probabilistic rectangular automaton, and M be the probabilistic multisingular automaton constructed from R. Let $\mathcal{S}_R$ and $\mathcal{S}_M$ be the concurrent probabilistic systems of R and M respectively. Let $s_1 \in S_M$ be a state of $\mathcal{S}_M$, and let $s_2 \in S_M$ be a state of $\mathcal{S}_R$ such that $s_1\mathcal{R}_f s_2$. Then, if there exists a continuous transition $s_1 \xrightarrow{\delta} s_1'$ of $\mathcal{S}_R$, then there exists a continuous transition $s_2 \xrightarrow{\delta} s_2'$ of $\mathcal{S}_M$ such that $s_1'\mathcal{R}_f s_2'$.*

Proof. Consider the transition of R of the the form $s_1 \xrightarrow{\delta} s_1'$. First we show the existence of a transition of M of the form $s_2 \xrightarrow{\delta} s'$ for *some* state $s' \in S_M$ (later we will show that there exists a target state of such a time transition which is in the relation $\mathcal{R}_f$ with s_1). Let $s_1 = (v, \mathbf{a})$. For each variable $x_i \in \mathcal{X}$, let $[l_i, u_i] = \mathit{flow}^R(v)_i$ and $[l_i', u_i'] = \mathit{inv}^R(v)_i$. Then $\mathit{lower}_i = \max\{l_i', \mathbf{a}_i + \delta.l_i\}$ is the minimum value that the variable x_i can take after δ time units have elapsed from state $(v, \mathbf{a})$, and $\mathit{upper}_i = \min\{u_i', \mathbf{a}_i + \delta.u_i\}$ is the corresponding maximal value. If we use $(v, \mathbf{b})$ to denote s_2, observe that $(v, \mathbf{a}) \in f(v, \mathbf{b})$, or, equivalently, $\mathbf{a} \in \prod_{i=1}^{n}[\mathbf{b}_{l(i)}, \mathbf{b}_{u(i)}]$. Now assume, by contradiction, that $(v, \mathbf{b}) \xrightarrow{\delta} s'$ does not exist; by the convexity of $\mathit{inv}^M(v)$, this means that s' is not an admissible state of M. More formally, if $s' = (v, \mathbf{d})$, then, for some variable $x_i \in \mathcal{X}$, we must have $\mathbf{d}_{l(i)} > u'_i$ or $\mathbf{d}_{u(i)} < l'_i$. Observe that $\mathbf{b}_{l(i)} \leq \mathbf{a}_i \leq \mathbf{b}_{u(i)}$, $\mathit{flow}^M(v)_{l(i)} = [l_i, l_i]$, $\mathit{flow}^M(v)_{u(i)} = [u_i, u_i]$, and that $\mathbf{d}_{l(i)}, \mathbf{d}_{u(i)}$ are defined as:

$$\begin{aligned}
\mathbf{d}_{l(i)} &= \max\{l'_i, \mathbf{b}_{l(i)} + \delta.l_i\} \\
\mathbf{d}_{u(i)} &= \min\{u'_i, \mathbf{b}_{u(i)} + \delta.u_i\} \ .
\end{aligned}$$

Consider $\mathbf{d}_{l(i)} > u'_i$; as $\mathbf{b}_{l(i)} \leq \mathbf{a}_i$, then

$$\mathbf{d}_{l(i)} = \max\{l'_i, \mathbf{b}_{l(i)} + \delta.l_i\} \leq \max\{l'_i, \mathbf{a}_{l(i)} + \delta.l_i\} = \mathit{lower}_i \ ,$$

and hence $\mathit{lower}_i > u'_i$. Similarly, from $\mathbf{d}_{u(i)} < l'_i$ we conclude that $\mathit{upper}_i < l'_i$. Then, in either of the two cases, there cannot exist the transition $s_1 \xrightarrow{\delta} s'_1$, which contradicts the assumption of the proposition. Therefore, there exists the transition $s_2 \xrightarrow{\delta} s'$ for some $s' \in S_M$.

This reasoning also allows us to show that the existence of a transition of M of the form $s_2 \xrightarrow{\delta} s'_2$ which is such that $s'_1 \mathcal{R}_f s'_2$. Let $\gamma_{(v,\mathbf{a})} \in \mathsf{Rect}(\mathcal{X})$ be the rectangle defined by

$$\gamma_{(v,\mathbf{a})} = \prod_{i=1}^{n} [\mathit{lower}_i, \mathit{upper}_i] \ .$$

Therefore, the rectangle $\gamma_{(v,\mathbf{a})}$ is the set of valuations that are possible after δ time units have elapsed from the state $(v, \mathbf{a})$ of $\mathcal{S}_R$. Let $s'_1 = (v, \mathbf{c})$. Hence $\mathbf{c} \in \gamma_{(v,\mathbf{a})}$. We now define a second rectangle $\gamma_{(v,\mathbf{b})} \in \mathsf{Rect}(\mathcal{X})$, which is the set of valuations that the δ-successor of $s_2 = (v, \mathbf{b})$, which we denote by $s'_2 = (v, \mathbf{d})$, represents via the function f, and show that $\gamma_{(v,\mathbf{a})} \subseteq \gamma_{(v,\mathbf{b})}$. Recall that $\mathbf{d}_{l(i)} \leq \mathit{lower}_i$ and $\mathbf{d}_{u(i)} \geq \mathit{upper}_i$. Clearly, by the definitions of $\mathbf{d}_{l(i)}, \mathbf{d}_{u(i)}$, we have $[\mathbf{d}_{l(i)}, \mathbf{d}_{u(i)}] \cap \mathit{inv}^R(v)_i = [\mathbf{d}_{l(i)}, \mathbf{d}_{u(i)}]$, and therefore $f(v, \mathbf{d}) = \{v\} \times (\prod_{i=1}^{n}[\mathbf{d}_{l(i)}, \mathbf{d}_{u(i)}] \cap \mathit{inv}^R(v)) = \{v\} \times \prod_{i=1}^{n}[\mathbf{d}_{l(i)}, \mathbf{d}_{u(i)}]$. Now let $\gamma_{(v,\mathbf{b})} = \prod_{i=1}^{n}[\mathbf{d}_{l(i)}, \mathbf{d}_{u(i)}]$. Then, from $\mathbf{d}_{l(i)} \leq \mathit{lower}_i \leq \mathit{upper}_i \leq \mathbf{d}_{u(i)}$ for each $1 \leq i \leq n$, we have $\gamma_{(v,\mathbf{a})} \subseteq \gamma_{(v,\mathbf{b})}$. Therefore $s'_1 = (v, \mathbf{c}) \in \{v\} \times \gamma_{(v,\mathbf{a})} \subseteq \{v\} \times \gamma_{(v,\mathbf{b})} = f(v, \mathbf{b}) = s'_2$, and therefore $s'_1 \mathcal{R}_f s'_2$. □

We illustrate the intuition underlying the proof in Figure 8.4. First we concentrate on the continuous transitions available in the state $(v, \mathbf{a})$ of R, which is shown on the left-hand side of Figure 8.4. The set of possible valuations that can be reached by R by letting δ time units elapse is shown by the dotted, inner rectangle on the right of the figure; this is the rectangle denoted by $\gamma_{(v,\mathbf{a})}$ in the proof of Lemma 8.4.4. We also show the maximal and minimal slopes that can be taken by continuous transitions from the state $(v, \mathbf{a})$ as dotted lines bordering the cone drawn from $(v, \mathbf{a})$ to $\gamma_{(v,\mathbf{a})}$.

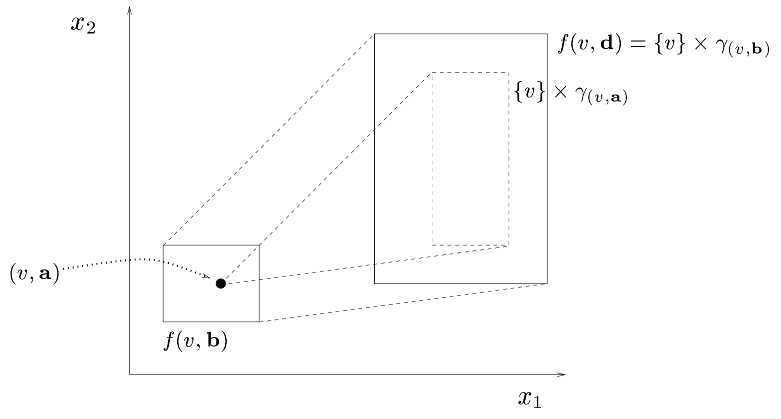

x_2
$f(v,\mathbf{d}) = \{v\} \times \gamma_{(v,\mathbf{b})}$
$\{v\} \times \gamma_{(v,\mathbf{a})}$
$(v,\mathbf{a})$
$f(v,\mathbf{b})$
x_1

Figure 8.4: An example illustrating Lemma 8.4.4.

Now consider the state $(v, \mathbf{b})$ of M, which encodes a "rectangular" set of states of R, denoted by $f(v, \mathbf{b})$, as shown on the left of Figure 8.4. As we consider the case in which $(v, \mathbf{a})\mathcal{R}_f(v, \mathbf{b})$, we have $(v, \mathbf{a}) \in f(v, \mathbf{b})$. Take the variable x_2, and consider its upper bound as encoded by the state $(v, \mathbf{b})$ of M, which is represented by the top horizontal border of the rectangle $f(v, \mathbf{b})$. As we are considering the upper bound of x_2, we are also concerned with the maximal rate of increase for this variable as δ time units elapse. Then the upper bound on the valuations of x_2 after such a continuous transition will be given by the top horizontal border of the solid, outer rectangle shown on the right-hand side of Figure 8.4. That is, this border encodes the maximal value of x_2, regardless of the precise value and rate of increase of x_1, provided that x_1 was within the bounds encoded by the original state $(v, \mathbf{b})$. As the maximal bound on x_2 encoded in the state $(v, \mathbf{b})$ must be greater than the value of x_2 in $(v, \mathbf{a})$, then the maximal bound on x_2 after δ time units elapse must be above the maximal value of x_2 that can be attained by $(v, \mathbf{a})$ after a continuous transition of the same duration; this is because, in both cases, the resulting values are obtained by taking the maximal rate of increase of x_2. We can repeat an analogous operation for the lower bound on x_2, taking the minimal rate of increase of x_2; then the lower bound on x_2 is encoded by the bottom horizontal border of the right-hand solid, outer rectangle. We can repeat this process for the variable x_1 to obtain the vertical borders of this rectangle. The state of M which encodes this set of states of R is denoted by $(v, \mathbf{d})$, and the set of states of R itself is denoted by $f(v, \mathbf{d}) = \{v\} \times \gamma_{(v,\mathbf{b})}$. Then it must be the case that $\gamma_{(v,\mathbf{a})} \subseteq \gamma_{(v,\mathbf{b})}$.

B – Discrete transitions

We now turn our attention to the two properties of discrete transitions that we require. Before these properties are stated formally, we introduce a preliminary definition which allows us to reason about the tuples of a probabilistic rectangular or multisingular automaton distribution that are used by a *concurrent probabilistic system distribution* to make a transition to a state. We note that the probabilistic rectangular automaton R used in the following definition need *not* be the one used elsewhere in this section; that is, the definition applies to both R of Lemma 8.4.3 *and* the constructed M, as any probabilistic multisingular automaton is a probabilistic rectangular automaton of a restricted form.

Definition 8.4.5 *Let R be a probabilistic rectangular automaton, and let $(v, \mathbf{a}) \in S_R$ be a state of its concurrent probabilistic system $\mathcal{S}_R$. If there exists a distribution $p \in prob(v)$ such that $\mathbf{a} \in pre_v(p)$, then consider a vector $\langle \mathbf{c} \rangle \in \mathsf{Combinations}(\mathbf{a}, \mathsf{extract}(p))$, and the transition $(v, \mathbf{a}) \xrightarrow{\theta_p, \tilde{p}_{\langle \mathbf{c} \rangle}}$. Then the function $\mathsf{switch}[\tilde{p}_{\langle \mathbf{c} \rangle}] : S_R \to 2^{\mathsf{support}(p)}$ is defined such that, for any state $(v', \mathbf{b}) \in S_R$, we have*

$$\mathsf{switch}[\tilde{p}_{\langle \mathbf{c} \rangle}](v', \mathbf{b}) = \{q^i \in \mathsf{support}(p) \mid \mathbf{b} = \mathbf{c}^i\} \ .$$

Therefore, the set $\mathsf{switch}[\tilde{p}_{\langle \mathbf{c} \rangle}](v', \mathbf{b})$ contains those tuples from $\mathsf{support}(p)$ which can be used to make the transition from the state $(v, \mathbf{a})$ to the state $(v', \mathbf{b})$, given the choice of the distribution $p \in prob(v)$ and vector $\langle \mathbf{c} \rangle \in \mathsf{Combinations}(\mathbf{a}, \mathsf{extract}(p))$. It can be seen that this definition applies to probabilistic multisingular automata, upon consideration of the fact that there will exist only *one* vector in the set of vectors given by $\mathsf{Combinations}(\mathbf{a}, \mathsf{extract}(p))$ (as post conditions of probabilistic multisingular automata contain a single point, rather than a set of points).

Recall that our aim is to prove two properties concerning the discrete transitions of $\mathcal{S}_R$ and $\mathcal{S}_M$.

B – Discrete transitions: part 1. We now present the lemma required for part 1 of the discrete transitions section. Note that we suppress vectors of the form $\langle \mathbf{c} \rangle \in \mathsf{Combinations}(\mathbf{a}, \mathsf{extract}(r))$, which usually appear as subscripts to distributions of the underlying concurrent probabilistic systems of probabilistic rectangular automata. That is, the notation $\tilde{r}$ refers to a distribution in which such a vector is implicit.

Lemma 8.4.6 (Discrete transitions: existence) *Let R be a probabilistic rectangular automaton, and M be the probabilistic multisingular automaton constructed from R. Let $\mathcal{S}_R$ and $\mathcal{S}_M$ be the concurrent probabilistic systems of R and M respectively. Let $s_1 \in S_M$ be a state of $\mathcal{S}_M$, and let $s_2 \in S_M$ be a state of $\mathcal{S}_R$ such that $s_1 \mathcal{R}_f s_2$. If there exists a discrete transition $s_1 \xrightarrow{\theta_r, \tilde{r}} s_1'$ of $\mathcal{S}_R$, then there exists a discrete transition $s_2 \xrightarrow{\theta_r, \tilde{m}} s_2'$ of $\mathcal{S}_M$ such that $s_1' \mathcal{R}_f s_2'$.*

Proof. Let $s_1 = (v, \mathbf{a})$, $s_2 = (v, \mathbf{b})$, $s_1' = (v', \mathbf{c})$, and $s_2' = (v', \mathbf{d})$. This proof will be divided into two sections:

- *Section A* gives a proof to show that the existence of a transition $(v, \mathbf{a}) \xrightarrow{\theta_r, \tilde{r}} (v', \mathbf{c})$ of $\mathcal{S}_R$ implies the existence of a transition $(v, \mathbf{b}) \xrightarrow{\theta_r, \tilde{m}} (v', \mathbf{d})$ of $\mathcal{S}_M$;
- *Section B* shows that $(v', \mathbf{c})\mathcal{R}_f(v', \mathbf{d})$.

Section A: existence of transitions. Recall that $r \in \mathit{prob}^R(v)$, which is featured in the subscript of the event θ_r, is the distribution of R from which the concurrent probabilistic system distribution $\tilde{r}$ is derived. From the construction of $\mathcal{S}_R$, the distribution r is enabled in the state $(v, \mathbf{a})$; that is, $\mathbf{a} \in \mathit{pre}^R_v(r)$. Also recall from the definition of M that there exists a set $\mathsf{derive}(r) \subseteq \mathit{prob}^M(v)$ of distributions of M that are derived from the single distribution r of R. Furthermore, recall the definition of their pre-conditions, and observe that these pre-conditions are disjoint; that is, for each variable $x_i \in \mathcal{X}$, either the values of both of $y_{l(i)}, y_{u(i)}$ are in $\mathit{pre}^M_v(m)$, or neither are, or one or the other of these variables is in this set. We show that one of these pre-conditions will be satisfied by the state $(v, \mathbf{b})$, recalling the fact that $(v, \mathbf{a})\mathcal{R}_f(v, \mathbf{b})$ is equivalent to $(v, \mathbf{a}) \in f(v, \mathbf{b})$. For each variable $x_i \in \mathcal{X}$, let $[l_i, u_i] = \mathit{pre}^R_v(r)_i$. From $\mathbf{a} \in \mathit{pre}^R_v(r)$, we have $l_i \leq \mathbf{a}_i \leq u_i$. Recall that $f(v, \mathbf{b}) = \{v\} \times \prod_{j=1}^n [\mathbf{b}_{l(j)}, \mathbf{b}_{u(j)}]$, and that $(v, \mathbf{a}) \in f(v, \mathbf{b})$ gives us the fact that $\mathbf{b}_{l(i)} \leq \mathbf{a}_i \leq \mathbf{b}_{u(i)}$. Therefore, from $l_i \leq \mathbf{a}_i \leq u_i$, it cannot be the case that either *both* $\mathbf{b}_{l(i)}, \mathbf{b}_{u(i)} < l_i$ or *both* $\mathbf{b}_{l(i)}, \mathbf{b}_{u(i)} > u_i$; by the construction of M, this is the *only* case in which *all* of the distributions in $\mathsf{derive}(r)$ are not enabled. Repeating this for all variables $x_i \in \mathcal{X}$ leads us to conclude that, from the construction of M, depending on the valuations $\mathbf{a}$ and $\mathbf{b}$, then a distribution m is enabled for choice in the state $(v, \mathbf{b})$. We let status be the status function with which m is associated.

Let $(v', \mathit{post}^R, X) \in \mathsf{support}(r)$ be a tuple in the support of r which is used to obtain the state $(v', \mathbf{c})$; that is, $(v', \mathit{post}^R, X) \in \mathsf{switch}[\tilde{r}](v', \mathbf{c})$. From the translation method from R to M, there must exist a tuple $(v', \mathit{post}^M, Y) \in \mathsf{support}(m)$ such that $(v', \mathit{post}^R, X) \rightsquigarrow (v', \mathit{post}^M, Y)$. We concentrate exclusively on these two tuples for the remainder of the proof, as they turn out to to be sufficient to show the required properties. Our first task is to show that $(v', \mathbf{d})$ is an admissible state. For each variable $x_i \in \mathcal{X}$, let $[l_i^{inv}, u_i^{inv}] = \mathit{inv}^R(v')_i$, and let $[l'_i, u'_i] = (\mathit{post}^R)_i$. Note that, by construction of M, we have $\mathit{inv}^M(v')_{l(i)} = (-\infty, u_i^{inv}]$, $\mathit{inv}^M(v')_{u(i)} = [l_i^{inv}, \infty)$, $(\mathit{post}^M)_{l(i)} = l'_i$, and $(\mathit{post}^M)_{u(i)} = u'_i$.

Consider the variable $x_i \in \mathcal{X}$. We consider the case in which $\mathsf{status}(x_i) = 1$, leaving the cases for the other values of the status function to the reader. There are two cases, depending on whether $x_i \in X$.

Case: $x_i \in X$. In this case, both $y_{l(i)}$ and $y_{u(i)}$ are reset to the values given by $(\mathit{post}^M)_{l(i)} = l'_i$ and $(\mathit{post}^M)_{u(i)} = u'_i$ respectively. We note, by the assumption of tightened post conditions of R (Definition 7.2.2) of R, we have $l'_i \geq l_i^{inv}$ and $u'_i \leq u_i^{inv}$. Then as $\mathbf{d}_{l(i)} = l'_i$ and $\mathbf{d}_{u(i)} = u'_i$, we have $l_i^{inv} \leq \mathbf{d}_{l(i)} \leq \mathbf{d}_{u(i)} \leq u_i^{inv}$.

Case: $x_i \notin X$**.** Then $(v', \mathbf{d})$ is such that $\mathbf{d}_{l(i)} = l_i$ (as the value of the variable $y_{l(i)}$ is reset to l_i), and $\mathbf{d}_{u(i)} = \mathbf{b}_{u(i)}$ (the value of the variable $y_{u(i)}$ is unchanged). By the assumption of tightened pre-conditions of R (Definition 7.2.3), we have $l_i \geq l_i^{inv}$ and $\mathbf{b}_{u(i)} \leq u_i^{inv}$. Note that, by the construction of M, we have $pre_v^M(m)_{u(i)} = (-\infty, u_i]$. That is, it is not possible for $\mathbf{b}_{u(i)} > u_i^{inv}$, as we know that $\mathbf{b}_{u(i)} \in (-\infty, u_i]$ by the assumption that $\mathsf{status}(x_i) = 1$ and the definition of the pre-condition of m. Then, because the assumption of tightened pre-conditions implies that, for any variable $x_i \notin X$ that is not reset on this transition, we have $pre_v^R(p)_i \subseteq inv^R(v')_i$; that is, $u_i \leq u_i^{inv}$. Hence, as $\mathbf{b}_{u(i)} = \mathbf{d}_{u(i)}$, we have $\mathbf{d}_{u(i)} \geq l_i^{inv}$, and we also have $\mathbf{d}_{l(i)} \leq u_i^{inv}$. Therefore $l_i^{inv} \leq \mathbf{d}_{l(i)} \leq \mathbf{d}_{u(i)} \leq u_i^{inv}$.

Repeating this for all variables $x_i \in \mathcal{X}$ gives us that $\mathbf{d} \in inv^M(v')$. Therefore, the existence of the transition $(v, \mathbf{a}) \xrightarrow{\theta_r, \tilde{r}} (v', \mathbf{c})$ implies the existence of the transition $(v, \mathbf{b}) \xrightarrow{\theta_r, \tilde{m}} (v', \mathbf{d})$.

Section B: showing that $(v', \mathbf{c})\mathcal{R}_f(v', \mathbf{d})$. The fact that the states $(v', \mathbf{c})$ and $(v', \mathbf{d})$ are obtained via the tuples $(v', post^R, X)$ and $(v', post^M, Y)$ which are such that $(v', post^R, X) \rightsquigarrow (v', post^M, Y)$ is integral to this part of the proof. We first return to the transition $(v, \mathbf{a}) \xrightarrow{\theta_r, \tilde{r}} (v', \mathbf{c})$, and identify the properties of the set of valuations $\gamma = \mathbf{a}[X := post^R]$ which, by definition, the valuation $\mathbf{c}$ must be contained within. For each variable $x_i \in X$, as $[l'_i, u'_i] = (post^R)_i$, we have $\gamma_i = \mathbf{a}[X := post^R]_i = [l'_i, u'_i]$; for every other variable $x_i \notin X$, we have $\gamma_i = \mathbf{a}[X := post^R]_i = [\mathbf{a}_i, \mathbf{a}_i]$. Then we can see that $\gamma = \prod_{i=1}^n \gamma_i$. Therefore, for *any* choice of vector in $\mathsf{Combinations}(\mathbf{a}, \mathsf{extract}(r))$, the target state resulting from choosing r nondeterministically, and then selecting the tuple $(v', post^R, X)$ probabilistically, will be a state $(v', \mathbf{c})$ such that $\mathbf{c} \in \gamma$.

We turn our attention to the target state of $(v, \mathbf{b})$ given that m and $(v', post^M, Y)$ have been chosen nondeterministically and probabilistically, respectively. Again, we consider a particular distribution $m \in \mathsf{derive}(r)$ with a status subscript such that $\mathsf{status}(x_i) = 1$ for all variables $x_i \in \mathcal{X}$, leaving the other cases to the reader. Our aim is to show that the valuation $\mathbf{d}$ is such that $\gamma \subseteq \prod_{i=1}^n [\mathbf{d}_{l(i)}, \mathbf{d}_{u(i)}]$; then, whatever the choice of vector from $\mathsf{Combinations}(\mathbf{a}, \mathsf{extract}(r))$, the target state $(v', \mathbf{c}) \in \{v'\} \times \gamma$ is *also* in the set $\{v'\} \times (\prod_{i=1}^n [\mathbf{d}_{l(i)}, \mathbf{d}_{u(i)}] \cap inv^R(v))$, and hence $(v', \mathbf{c}) \in f(v', \mathbf{d})$. Consider the variable $x_i \in \mathcal{X}$; as in the previous section, there are two cases, depending on whether $x_i \in X$ or $x_i \notin X$.

Case: $x_i \in X$**.** Then, by construction of M, both variables $y_{l(i)}, y_{u(i)}$ are reset to the values of the interval $(post^R)_i = [l'_i, u'_i]$. Therefore, $\mathbf{d}_{l(i)} = l'_i$ and $\mathbf{d}_{u(i)} = u'_i$. Clearly, $\gamma_i = [\mathbf{d}_{l(i)}, \mathbf{d}_{u(i)}]$.

Case: $x_i \notin X$**.** By definition of M as a translation of R, the value of $y_{l(i)}$ is reset to l_i, while the value of $y_{u(i)}$ remains the same; that is, we have $\mathbf{d}_{l(i)} = l_i$ and

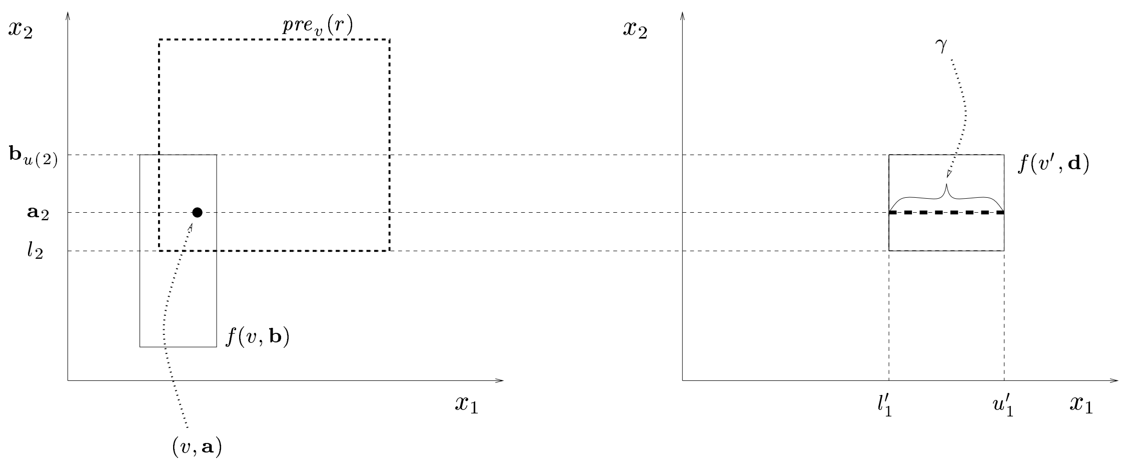


Figure 8.5: An example illustrating Lemma 8.4.6.

$\mathbf{d}_{u(i)} = \mathbf{b}_{u(i)}$. From the fact that $(v, \mathbf{a}) \in f(v, \mathbf{b})$, we have $\mathbf{a}_i \in [\mathbf{b}_{l(i)}, \mathbf{b}_{u(i)}]$, and, more pertinently, $\mathbf{a}_i \leq \mathbf{b}_{u(i)}$. Furthermore, as $\mathbf{a} \in pre^R_v(r)$, then $\mathbf{a}_i \in [l_i, u_i]$, and therefore $\mathbf{a}_i \geq l_i$. Hence $[\mathbf{a}_i, \mathbf{a}_i] = \gamma_i \subseteq [\mathbf{d}_{l(i)}, \mathbf{d}_{u(i)}]$.

Repeating this for all variables $x_i \in \mathcal{X}$, it must be the case that $\gamma \subseteq \prod_{i=1}^n [\mathbf{d}_{l(i)}, \mathbf{d}_{u(i)}]$ and, by the arguments of *Section A*, that $\prod_{i=1}^n [\mathbf{d}_{l(i)}, \mathbf{d}_{u(i)}] \subseteq inv^R(v)$. Hence, as $\mathbf{c} \in \gamma$, then we have $\mathbf{c} \in (\prod_{i=1}^n [\mathbf{d}_{l(i)}, \mathbf{d}_{u(i)}] \cap inv^R(v))$. Then it follows naturally that $(v', \mathbf{c}) \in \{v'\} \times (\prod_{i=1}^n [\mathbf{d}_{l(i)}, \mathbf{d}_{u(i)}] \cap inv^R(v))$. Therefore, $(v', \mathbf{c}) \in f(v', \mathbf{d})$, and hence $(v', \mathbf{c}) \mathcal{R}_f (v', \mathbf{d})$. $\square$

The intuition underlying this result is shown in Figure 8.5. The left-hand part of the diagram shows the state $(v, \mathbf{a})$ of R, and the set of states $f(v, \mathbf{b})$ of R encoded by a single state $(v, \mathbf{b})$ of M, before the discrete transition, whereas the right-hand side illustrates state sets of R after the transition. First consider the left-hand side. The bold dotted rectangle represents the pre-condition $pre_v(r)$ of the distribution r; naturally, the state $(v, \mathbf{a})$ is contained within this set. This state is also contained within the set of states of R denoted by $f(v, \mathbf{b})$, which is the set that the state $(v, \mathbf{b})$ of M encodes. The set $f(v, \mathbf{b})$ is illustrated by the solid rectangle. The lower bounds on the values of x_1 and x_2 featured in the set $f(v, \mathbf{b})$ are below the lower bounds on the set $pre_v(r)$, which corresponds to the case in which the transition r of R is represented by the distribution m of M for which the status function of m, denoted by status, is such that $\mathsf{status}(x_1) = \mathsf{status}(x_2) = 1$.

Now consider the effects of a discrete transition. First we identify the set of states that the state of R must be contained within after the transition. Say that the tuple of r chosen to make the transition from $(v, \mathbf{a})$ is such that the value of x_1 should be reset within the interval $[l'_1, u'_1]$, whereas the value of x_2 should remain constant. Then the value of x_1 may be reset nondeterministically to any value between l'_1 and u'_1, depending on the vector chosen from the set $\mathsf{Combinations}(\mathbf{a}, \mathsf{extract}(r))$. Hence the possible states which can be reached by R after the transition from $(v, \mathbf{a})$ are given by the bold horizontal dotted line labelled by γ on the right-hand side of Figure 8.5. Now consider the effects of a transition of M from $(v, \mathbf{b})$ using a tuple derived from the tuple of r used to make the transition of R. As the lower and upper bounds on the value of x_1 are represented by l'_1 and u'_1 respectively, these bounds are encoded in the state $(v', \mathbf{d})$ of M. As the value of x_2 is unchanged after the transition, the upper bound on its value remains at $\mathbf{b}_{u(2)}$. However, the lower bound on x_2 is updated to l_2 to reflect the lower bound on the pre-condition $pre_v(r)$. Therefore, the "rectangular" set of states $f(v', \mathbf{d})$ is given by the solid rectangle on the right-hand side of Figure 8.5. This rectangle contains the set γ, as the x_1 coordinates of $f(v', \mathbf{d})$ are equal to those of γ, and, with regard to x_2, the value of $\mathbf{a}_2$ must be above the lower bound on x_2 required for the satisfaction of the pre-condition $pre_v(r)$, and also below $\mathbf{b}_{u(2)}$ from $(v, \mathbf{a}) \in f(v, \mathbf{b})$.

B – Discrete transitions: part 2. We now show that, given the distributions $\tilde{r}$ and $\tilde{m}$ of $\mathcal{S}_R$ and $\mathcal{S}_M$ respectively, which are identified in part 1 of this section on discrete transitions, then there exists a weight function for $(\tilde{r}, \tilde{m})$ with respect to $\mathcal{R}_f$. Our proof is based on the relationship between the tuples in the support of r and m, and the states in the support of $\tilde{r}$ and $\tilde{m}$ that they are used to derive. To obtain some intuition concerning this relationship, consider the following example. Recall that the set of tuples of a distribution r of R are used to obtain a set of tuples of a distribution $m \in \mathsf{derive}(r)$ of M, where m is constructed to represent r in the manner described in the translation method given above. Then it is clear from the definition of M that each tuple in the support of r is used to obtain only one tuple of m; however, a tuple of m may be obtained from more than one tuple of r. Now consider the distributions $\tilde{r}$ and $\tilde{m}$ of the concurrent probabilistic systems $\mathcal{S}_R$ and $\mathcal{S}_M$ respectively, where $\tilde{r}$ is derived from r and $\tilde{m}$ is derived from m. An example of the relationship between the tuples in the support of r and m, and of the states in the support of $\tilde{r}$ and $\tilde{m}$, is shown in Figure 8.6. That is, $\mathsf{support}(r) = \{q_1, q_2, q_3\}$, $\mathsf{support}(m) = \{q'_1, q'_2\}$, $\mathsf{support}(\tilde{r}) = \{s_1, s_2\}$, and $\mathsf{support}(\tilde{m}) = \{s'_1, s'_2\}$. For each distribution, the interval [0,1] is given, denoting the probability assigned to the relevant element in its support.

Consider the state s_1 in the support of $\tilde{r}$; then this state is obtained from the two tuples q_1, q_2 in the support of r, and its probability is computed by summing the probability of these tuples. Then from Definition 8.4.5, we have $\mathsf{switch}[\tilde{r}](s_1) = \{q_1, q_2\}$. Now consider the state s'_2 in the support of $\tilde{m}$, which is obtained from the single tuple of m denoted by q'_2. Then, from Definition 8.4.5, it is clear that $\mathsf{switch}[\tilde{m}](s'_2) = \{q'_2\}$.

We now introduce two further definitions to reason about the tuples in the support of r and m.

Definition 8.4.7 *Let R be a probabilistic rectangular automaton, and let M the prob-*

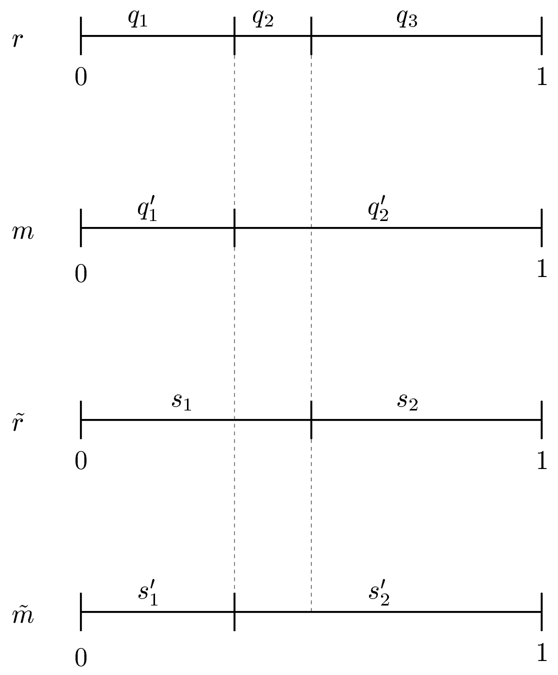


Figure 8.6: An example of the relationship between r, m, $\tilde{r}$ and $\tilde{m}$.

abilistic multisingular automaton that is constructed from R. Let $r \in prob^R(v)$ be a distribution of R, for some location $v \in V$, and let $m \in \mathsf{derive}(r)$ be a distribution of M derived from r. Then let $\mathsf{sw2sw} : 2^{\mathsf{support}(m)} \to 2^{\mathsf{support}(r)}$ be the function such that, for any set $Q \subseteq \mathsf{support}(m)$ of tuples, then $\mathsf{sw2sw}(Q) = \{q \in \mathsf{support}(r) \mid \exists q' \in Q$ such that $q \rightsquigarrow q'\}$.

That is, for any subset Q of tuples from the support of a distribution m of M, then $\mathsf{sw2sw}(Q)$ is the set of tuples of r from which the tuples in Q were obtained. In the example given above, observe that the tuple q'_2 in the support of the distribution m is obtained from *two* tuples in the support of the distribution r, namely q_2 and q_3. Then, by Definition 8.4.7, we have $\mathsf{sw2sw}(q'_2) = \{q_2, q_3\}$. Then it follows that we can identify the tuples of the distribution r which the state s'_2 *in the support of $\tilde{m}$ is derived* using the information encoded in $\mathsf{sw2sw}(q'_2)$; that is, we can let $\mathsf{sw2sw}(\mathsf{switch}[\tilde{m}](s'_2)) = \{q_2, q_3\}$ be the tuples of r which are used to obtain the tuples of m from which the state s'_2 is derived. Intuitively, we use the functions $\mathsf{switch}[\tilde{m}]$ and $\mathsf{sw2sw}$ to identify the tuples in the support of r which a state in the support of $\tilde{m}$ is *originally* derived from, via at least one tuple in the support of m.

We now define the final function necessary for this section.

Definition 8.4.8 *Let r, S_R, S_M, $\tilde{r}$ and $\tilde{m}$ be defined as in Lemma 8.4.6. The function $\mathsf{intersect}[\tilde{r}, \tilde{m}] : S_R \times S_M \to 2^{\mathsf{support}(r)}$ is defined such that, for the states $s'_1 \in S_R$, $s'_2 \in S_M$, we have $\mathsf{intersect}[\tilde{r}, \tilde{m}](s'_1, s'_2) = \mathsf{switch}[\tilde{r}](s'_1) \cap \mathsf{sw2sw}(\mathsf{switch}[\tilde{m}](s'_2))$.*

Intuitively, the function $\mathsf{intersect}[\tilde{r}, \tilde{m}]$ returns the tuples of r which are used to obtain a state s_1' of $\mathcal{S}_R$ and, indirectly via the distribution m, the state $s_2' \in S_M$. In the example above, to compute $\mathsf{intersect}[\tilde{r}, \tilde{m}](s_1', s_2')$, we take the intersection of the relevant sets $\mathsf{switch}[\tilde{r}](s_1')$ and $\mathsf{sw2sw}(\mathsf{switch}[\tilde{m}](s_2'))$ to obtain $\{q_2\}$. That is, the tuple q_2 of the distribution r is used to obtain both the state s_1 of R and, via conversion to a tuple of m, the state s_2' of M.

We now proceed to show the importance of the above definitions for the identification of a weight function for distributions such as $\tilde{r}$ and $\tilde{m}$ with respect to $\mathcal{R}_f$.

Lemma 8.4.9 (Discrete transitions: weight function) *Let $\tilde{r}$ and $\tilde{m}$ be defined as in Lemma 8.4.6. Then there exists a weight function for $(\tilde{r}, \tilde{m})$ with respect to $\mathcal{R}_f$.*

Proof. As the distributions $\tilde{r}$ and $\tilde{m}$ are fixed, we write $\mathsf{intersect}$ for $\mathsf{intersect}[\tilde{r}, \tilde{m}]$ for the remainder of this proof. Let $w : S_R \times S_M \to [0,1]$ be a function defined in the following way: for $s \in S_R$, $s' \in S_M$, simply let

$$w(s, s') = \sum_{q \in \mathsf{intersect}(s,s')} r(q) \ .$$

We show that w is the required weight function. Recall the conditions included in the definition of weight functions (Definition 3.2.12). First, we show that $w(s,s') > 0$ implies that $s \mathcal{R}_f s'$. Note that $w(s,s') > 0$ implies that there exists a tuple $q \in \mathsf{intersect}(s,s')$, which in turn implies that $q \in \mathsf{switch}[\tilde{r}](s)$ and $q \in \mathsf{sw2sw}(\mathsf{switch}[\tilde{m}](s'))$. The latter implies that there exists a tuple $q' \in \mathsf{support}(m)$ such that $q \in \mathsf{sw2sw}(q')$, from which we conclude that $q \rightsquigarrow q'$. From reasoning given in *Section B* of the proof of Lemma 8.4.6, it follows that $s \mathcal{R}_f s'$.

Our next task to show that w is the required weight function concerns showing that

$$\sum_{s'' \in S_M} w(s, s'') = \tilde{r}(s) \ , \ \sum_{s'' \in S_R} w(s'', s') = \tilde{m}(s')$$

for any $s \in S_R$, $s' \in S_M$. We consider the first equation, with the case for the second being symmetric. Note that, for any $s \in S_R$, by the definition of w given above:

$$\sum_{s'' \in S_M} w(s, s'') = \sum_{s'' \in S_M} \sum_{q \in \mathsf{intersect}(s,s'')} r(q) \ .$$

From this fact, and as $\mathsf{switch}[\tilde{r}](s) = \emptyset$ for any $s \in S_R \setminus \mathsf{support}(\tilde{r})$, and $\mathsf{switch}[\tilde{m}](s') = \emptyset$ for any $s' \in S_M \setminus \mathsf{support}(\tilde{m})$, we can reduce our problem to that of showing

$$\sum_{s'' \in \mathsf{support}(\tilde{m})} \sum_{q \in \mathsf{intersect}(s,s'')} r(q) = \tilde{r}(s) \tag{8.1}$$

for any $s \in \mathsf{support}(\tilde{r})$. Now consider the right-hand side of this equation, and observe that

$$\tilde{r}(s) = \sum_{q \in \mathsf{switch}[\tilde{r}](s)} r(q) \tag{8.2}$$

for some $s \in \mathsf{support}(\tilde{r})$, by the definition of $\mathsf{switch}[\tilde{r}]$ (Definition 8.4.5). Putting Equation 8.1 and Equation 8.2 together, our problem then reduces to showing that

$$\sum_{s'' \in \mathsf{support}(\tilde{m})} \sum_{q \in \mathsf{intersect}(s,s'')} r(q) = \sum_{q \in \mathsf{switch}[\tilde{r}](s)} r(q) \ .$$

Now, by Lemma 3.1, this problem can in turn be reduced to that of showing that the set of sets $\{\mathsf{intersect}(s,s'') \mid s'' \in \mathsf{support}(\tilde{m})\}$ is an $\emptyset$-inclusive partition of $\mathsf{switch}[\tilde{r}](s)$.

Firstly, to prove that the elements of the set $\{\mathsf{intersect}(s,s'') \mid s'' \in \mathsf{support}(\tilde{m})\}$ are disjoint, we observe that $\mathsf{intersect}(s,s''_1) \cap \mathsf{intersect}(s,s''_2) = \emptyset$ for any distinct $s''_1, s''_2 \in \mathsf{support}(\tilde{m})$, because of the fact that $\mathsf{sw2sw}(\mathsf{switch}[\tilde{m}](s''_1)) \cap \mathsf{sw2sw}(\mathsf{switch}[\tilde{m}](s''_2)) = \emptyset$ (as it is not possible to derive two different tuples of m from the same tuple of r), which in turn follows from the fact that $\mathsf{switch}[\tilde{m}](s''_1) \cap \mathsf{switch}[\tilde{m}](s''_2) = \emptyset$ (as it is not possible to derive two different states in the support of $\tilde{m}$ from the same tuple of m).

Secondly, to show that $\bigcup_{s'' \in \mathsf{support}(\tilde{m})} \mathsf{intersect}(s,s'') = \mathsf{switch}[\tilde{r}](s)$, we observe the following:

$$\begin{aligned}
& \bigcup_{s'' \in \mathsf{support}(\tilde{m})} \mathsf{intersect}(s,s') \\
= & \bigcup_{s'' \in \mathsf{support}(\tilde{m})} (\mathsf{switch}[\tilde{r}](s) \cap \mathsf{sw2sw}(\mathsf{switch}[\tilde{m}](s''))) && \text{(by definition of } \mathsf{intersect}) \\
= & \ \mathsf{switch}[\tilde{r}](s) \cap \bigcup_{s'' \in \mathsf{support}(\tilde{m})} \mathsf{sw2sw}(\mathsf{switch}[\tilde{m}](s'')) \\
= & \ \mathsf{switch}[\tilde{r}](s) \cap \mathsf{sw2sw}(\bigcup_{s'' \in \mathsf{support}(\tilde{m})} \mathsf{switch}[\tilde{m}](s'')) \\
= & \ \mathsf{switch}[\tilde{r}](s) \cap \mathsf{sw2sw}(\mathsf{support}(m)) \\
& && \text{(by } \textstyle\bigcup_{s'' \in \mathsf{support}(\tilde{m})} \mathsf{switch}[\tilde{m}](s'') = \mathsf{support}(m)) \\
= & \ \mathsf{switch}[\tilde{r}](s) \cap \mathsf{support}(r) \\
& && \text{(by } \mathsf{sw2sw}(\mathsf{support}(m)) = \mathsf{support}(r)) \\
= & \ \mathsf{switch}[\tilde{r}](s) && \text{(by } \mathsf{switch}[\tilde{r}](s) \subseteq \mathsf{support}(r)).
\end{aligned}$$

This concludes the proof of showing that w is a weight function for $(\tilde{r}, \tilde{m})$ with respect to $\mathcal{R}_f$. □

C – Simulation

We are now in a position to prove Lemma 8.4.3, and recall the lemma:

Lemma 8.4.3 *Let R be a probabilistic rectangular automaton, and M be the probabilistic multisingular automaton constructed from R. Let $\mathcal{S}_R$ and $\mathcal{S}_M$ be the concurrent probabilistic systems of R and M respectively. Let $s_1 \in S_M$ be a state of $\mathcal{S}_M$, and let $s_2 \in f(s)$ be a state of R. Then $s_1 \preceq s_2$.*

Proof. Follows from Lemma 8.4.4 and Lemma 8.4.6. Note that continuous transitions are performed according to Dirac distributions, and that, for any states $s \in S_R$, $s' \in S_M$, we have $\mathcal{D}(s)\mathcal{R}_f\mathcal{D}(s')$ if and only if $s\mathcal{R}_f s'$. □

Proposition 8.4.10 *Let $\mathcal{S}_R$ and $\mathcal{S}_M$ be the concurrent probabilistic systems of the probabilistic rectangular automaton R and the probabilistic multisingular automaton M constructed from R, respectively. For any $\forall PBTL$ formula Φ, if $\mathcal{S}_M \models_{\mathcal{A}} \Phi$ then $\mathcal{S}_R \models_{\mathcal{A}} \Phi$.*

Proof. Follows immediately from Theorem 3.3.5 and Lemma 8.4.3 □

Chapter 9

Reachability for probabilistic hybrid automata

9.1 Introduction

Although the methods for model checking probabilistic rectangular automata by using translations to probabilistic timed automata can be used for the verification of a wide class of probabilistic temporal logic properties, they suffer from two significant drawbacks. Firstly, the methods require a number of *restrictive assumptions* to be placed on the model; in particular, that of initialisation. Secondly, the translation from probabilistic stopwatch automata to the timed model introduces additional locations, and, more seriously, the translation from probabilistic rectangular automata to the multisingular model doubles the number of variables. Because of the resulting increase in the state-space of the system, these methods can be regarded as being *inefficient*, and therefore impractical from the point of view of performing verification on real-life systems.

Therefore, we introduce a method for the verification of probabilistic rectangular automata against *probabilistic reachability* properties, which, while less expressive than the properties of PBTL, permit the use of a *direct* analysis of the reachable state space of the model. The information obtained by such an analysis can then be used to compute the reachability probability of the appropriate target set. Our method is an extension of the forward reachability method of Chapter 6; we now deal with models for which variables can increase at rates not necessarily the same as real-time, and for which variables may be set to values other than 0. Therefore, our first task is to redefine the post operation accordingly, noting that the operand and result of post are both convex *polyhedra*, rather than simply zones as in the timed case in Chapter 6. The forward reachability algorithm **FRRectangular**, and the definition of the forward reachability graph in this context, follow in a natural manner from the probabilistic timed automaton case (see Figure 6.1 and Definition 6.3.3).

However, the final task, which involves the proof of correctness of our approach, is more involved than the similar result in Chapter 6; the useful characteristics of zones and resets in Section 6.6.1, which permitted the relatively simple proof of correctness of Proposition 6.6.6, are lost by the move to probabilistic rectangular automata. However, we can nevertheless make use of ideas taken from the characterisation of the simulation preorder of Definition 3.2.15 to obtain the required result.

9.2 Preliminaries

First we extend the notion of a symbolic state from the timed to rectangular case. In particular, the natural way to represent sets of valuations of probabilistic rectangular automata is as convex polyhedra, as the generality of the continuous transitions involved in rectangular automata requires a more general representation mechanism for time successors state sets than zones.

Definition 9.2.1 (Symbolic states (rectangular case)) *For the probabilistic rectangular automaton R, a pair of the form (v, η), where $v \in V$ and $\eta \in \mathsf{ConvexPoly}(\mathcal{X})$ is a convex polyhedron such that $\eta \subseteq inv(v)$, is called a* symbolic state*.*

Recall the definition of forward projection for zones $\nearrow\zeta$ (see Section 3.4). We extend this definition to reflect the fact that, as time passes during the execution of a probabilistic rectangular automaton, the continuous dynamics of the variables are governed by a rectangle $\gamma \in \mathsf{Rect}(\mathcal{X})$, rather than just by the single valuation $[1,1]^n$, as in the timed case.

Definition 9.2.2 (Forward projection (rectangular case)) *For the rectangle $\gamma \in \mathsf{Rect}(\mathcal{X})$ and convex polyhedron $\eta \in \mathsf{Poly}(\mathcal{X})$, the forward projection operator $\nearrow[\gamma]\eta$ is defined as*

$$\nearrow[\gamma]\eta \stackrel{\text{def}}{=} \{\mathbf{a} \in \mathbb{R}^n \mid \exists \mathbf{f} \in \gamma . \exists \delta \in \mathbb{R}_{\geq 0} . \mathbf{a} - \delta . \mathbf{f} \in \eta\} .$$

The intuitive meaning of $\nearrow[\gamma]\eta$ is as follows: the convex polyhedron $\nearrow[\gamma]\eta$ represents a set of valuations, where each valuation can be obtained from a valuation in the convex polyhedron γ by selecting a gradient from γ and letting some amount of time pass (note that δ represents time). Observe that $\nearrow[[1,1]^n]\zeta = \nearrow\zeta$ for any zone $\zeta \in \mathsf{Zone}(\mathcal{X})$.

An *edge* of a probabilistic rectangular automaton is denoted by a tuple of the form $e = (v, v', post, X, p)$, where $v, v' \in V$, $p \in prob(v)$ and $p(v', post, X) > 0$. The set $\mathsf{out}(v)$ denotes all of the edges of the form $(v, \_, \_, \_, \_)$.

We now proceed to define the successor operations for probabilistic rectangular automata.

Definition 9.2.3 (Successor operations (rectangular case)) *Let (v, η) be a symbolic state of the probabilistic rectangular automaton R. Then:*

Time successor *The* time successor *of (v, η) is defined as*

$$\mathsf{time\_succ}(v, \eta) \stackrel{\text{def}}{=} (v, \nearrow[\mathit{flow}(v)]\eta \cap \mathit{inv}(v)) \ .$$

Discrete successor *The* discrete successor *of (v, η) with respect to the edge $e = (v, v', \mathit{post}, X, p)$ is defined as*

$$\mathsf{disc\_succ}(e, (v, \eta)) \stackrel{\text{def}}{=} (v', ((\eta \cap \mathit{pre}_v(p))[X := \mathit{post}]) \cap \mathit{inv}(v')) \ .$$

Post *For an edge $e \in \mathsf{out}(v)$, the* post operation *is defined according to:*

$$\mathsf{post}[e](v, \zeta) \stackrel{\text{def}}{=} \mathsf{time\_succ}(\mathsf{disc\_succ}(e, (v, \zeta))) \ .$$

Observe that $\nearrow[\mathit{flow}(v)]\eta$ denoted the convex polyhedron which contains all of the valuations that can be reached from any valuation in η by letting time elapse, according to any gradient in the rectangle $\mathit{flow}(v)$. Also note that, unlike the definition of the post operation in Definition 6.3.2, there is no c-closure operation involved in the definition of this operation above. Recall that the c-closure operation guaranteed that the forward reachability algorithm from this previous chapter terminated, as, for any finite $c \in \mathbb{N}$, there are finitely many c-closed zones. We do not include such an operation here, as the reachability problem for non-initialised, non-probabilistic stopwatch automata is undecidable [HKPV98].

9.3 Forward reachability algorithm

Throughout this section, we fix the target set $\mathit{Target} \subseteq V$ of locations. As the forward reachability algorithm **FRRectangular** is similar to the algorithm **ForwardReachability** shown in Figure 6.1, we only highlight the difference between the former algorithm and the latter. Firstly, we do not require the constant c, as it was used in the c-closure component of the post operation; therefore, the line which includes its initialisation can be omitted. Secondly, the set *Fringe* initially comprises the single element $\mathsf{time\_succ}(v, \mathit{init}(v))$, where $v \in V$ is the unique location for which $\mathit{init}(v) \neq \emptyset$. Finally, the operation post does not include the constant c as a parameter, and instead is defined as in Definition 9.2.3.

As explained in Section 9.2, it is not guaranteed that the algorithm **FRRectangular** will terminate when applied even to probabilistic stopwatch automata. Because this characteristic is inherited from the non-probabilistic context, we also note that such forward reachability algorithms for non-probabilistic stopwatch automata (and therefore their superclasses) may not terminate. However, the claim of [HHW97], which asserts that there is no practical difference between the case in which verification exhausts resources (whether time or memory) and the case in which the algorithm will not terminate, also applies to the forward reachability algorithm presented here. Henceforth, given the probabilistic rectangular automaton R, and the target set $\mathit{Target} \subseteq V$, we assume that the forward reachability algorithm **FRRectangular** terminates.

The sets *SymbStates* and *Reached* obtained using the algorithm **FRRectangular** are now used to define the forward reachability graph for R with respect to *Target*, which takes the form of the concurrent probabilistic system $\mathcal{S}_R^{\mathit{Target}}$.

Definition 9.3.1 (Forward reachability graph (rectangular case)) *The forward reachability graph for the probabilistic rectangular automaton R and the target set $\mathit{Target} \subseteq V$, is the concurrent probabilistic system $\mathcal{S}_R^{\mathit{Target}} = (\langle S \rangle, \langle \bar{s} \rangle, \langle L \rangle, \langle \Sigma \rangle, \langle \mathit{Steps} \rangle)$ defined in the following way.*

- *The state set is the set $\langle S \rangle = \mathit{SymbStates}$.*
- *The initial state $\langle \bar{s} \rangle \in \langle S \rangle$ is the symbolic state $\mathsf{time\_succ}(\bar{v}, \mathit{init}(\bar{v})))$, where $v \in V$ is the single location for which $\mathit{init}(v) \neq \emptyset$.*
- *The labelling function $\langle L \rangle$ is arbitrary.*
- *The event set is $\langle \Sigma \rangle = \{\theta_p \mid v \in V, p \in \mathit{prob}(v)\} \cup \{\perp\}$.*
- *The transition relation $\langle \mathit{Steps} \rangle : \langle S \rangle \to \langle \Sigma \rangle \times \mu(\langle S \rangle)$ is characterised in the following way. For each symbolic state $(v, \zeta) \in \langle S \rangle$:*
  - *if $(v, \zeta) \in \mathit{SymbStates} \setminus \mathit{Reached}$, then for each distribution $p \in \mathit{prob}(v)$, where $\zeta \cap \mathit{pre}_v(p) \neq \emptyset$, there exists $(\theta_p, \hat{p}) \in \langle \mathit{Steps} \rangle(v, \zeta)$ such that, for each $(v', \zeta') \in \langle S \rangle$:*

$$\hat{p}(v', \zeta') = \sum_{\substack{X \subseteq \mathcal{X} \; \& \\ (v', \zeta') = \mathsf{post}[(v, v', \mathit{post}, X, p), c](v, \zeta)}} p(v', \mathit{post}, X) \; .$$

  - *if $(v, \zeta) \in \mathit{Reached}$, let $\langle \mathit{Steps} \rangle(v, \zeta) = \{(\perp, \mathcal{D}(v, \zeta))\}$.*

We then solve the probabilistic reachability problem $(\mathit{Target}, \sqsupseteq, \lambda)$ for the probabilistic rectangular automaton R by reduction to the maximal reachability problem for concurrent probabilistic systems, the solution to which is evaluated on the forward reachability graph $\mathcal{S}_R^{\mathit{Target}}$. That is, if we let $\Pr(\langle \bar{s} \rangle)$ be the maximal reachability probability using the methods of, for example, [dA97a], computed for the state $\langle \bar{s} \rangle$ with respect to the set of destination states *Reached*, then the answer to the probabilistic reachability problem is:

- "Maybe", if *Reached* $\neq \emptyset$ and $\Pr(\langle \bar{s} \rangle) \sqsupseteq \lambda$, and
- "No", otherwise.

9.4 Proof of correctness

We now prove the correctness of the forward reachability method presented in Section 9.3. First, we focus on the properties of individual transitions of the concurrent probabilistic system of the probabilistic rectangular automaton in question, and observe how they relate to the transitions of the forward reachability graph. Then, we use these results to show properties of adversaries to obtain the required result.

9.4.1 Properties of single-step transitions

We now formally state the required lemma concerning the properties of transitions both of the concurrent probabilistic system of a probabilistic rectangular automata, and of its associated forward reachability graph.

Lemma 9.4.1 *Let R be a probabilistic rectangular automaton R with the associated concurrent probabilistic system $\mathcal{S}_R$, let Target $\subseteq V$ be a target set of locations, and let $\mathcal{S}_R^{Target}$ be the resulting forward reachability graph. Then, for any symbolic state $(v, \eta) \in \langle S \rangle$ such that $v \notin$ Target, state $(v, \mathbf{a}) \in S_R$ such that $(v, \mathbf{a}) \in (v, \eta)$, and transition $(v, \mathbf{a}) \xrightarrow{\delta} \xrightarrow{\theta_p, \tilde{p}_{\langle \mathbf{b} \rangle}}$ of $\mathcal{S}_R$, there exists a transition $(v, \eta) \xrightarrow{\theta_p, \hat{p}}$ of $\mathcal{S}_R^{Target}$, such that $\tilde{p} \in \hat{p}$.*

Proof. The proof consists of two of sub-tasks. Firstly, we show that, given the transition $(v, \mathbf{a}) \xrightarrow{\delta} \xrightarrow{\theta_p, \tilde{p}_{\langle \mathbf{b} \rangle}}$, we can find a transition $(v, \eta) \xrightarrow{\theta_p, \hat{p}}$ for some distribution $\hat{p} \in \mu(\langle S \rangle)$; we say that this task concerns *enabledness*. Secondly, we show that there exists a weight function for $(\tilde{p}_{\langle \mathbf{b} \rangle}, \hat{p})$ with respect to $\in$, which means that $\tilde{p}_{\langle \mathbf{b} \rangle} \in \hat{p}$; we say that this task concerns the definition of a *weight function*.

Enabledness. We first show that the existence of the appropriate transition follows from the definitions of the operator post and the forward reachability graph for R with respect to *Target* $\subseteq V$ (Definition 9.2.3 and Definition 9.3.1, respectively). Recall the definition of post, and the definition of the single initial state $\langle \bar{s} \rangle$ of $\mathcal{S}_R^{Target}$ as $\mathsf{time\_succ}(\bar{v}, init(\bar{v}))$ for the appropriate initial location $\bar{v} \in V$ for which $init(\bar{v}) \neq \emptyset$. From this fact, and by the definition of post, we observe that, for any state $s \in S_R$ contained in the symbolic state $s' \in \langle S \rangle$, all of the flow-successors of s are also in s'.

The existence of the transition $(v, \mathbf{a}) \xrightarrow{\delta} \xrightarrow{\theta_p, \tilde{p}_{\langle \mathbf{b} \rangle}}$ of $\mathcal{S}_R$ implies that there exists $\mathbf{f} \in$ *flow*(v) such that $\mathbf{a} + \delta.\mathbf{f} \in pre_v(p)$ for the distribution $p \in prob(v)$. As $\mathbf{a} + \delta.\mathbf{f} \in \eta$, by the fact that all flow-successors of $(v, \mathbf{a})$ are in (v, η), then $\eta \cap pre_v(p) \neq \emptyset$. Therefore, by the definition of the transition relation $\langle \mathit{Steps} \rangle$ of $\mathcal{S}_R^{\mathit{Target}}$ (see Definition 9.3.1), there exists a transition $(v, \eta) \xrightarrow{\theta_p, \hat{p}}$ for some $\hat{p} \in \mu(\langle S \rangle)$. This completes the first task.

Weight function. Our second task is to show that there exists a weight function for $(\tilde{p}, \hat{p})$ with respect to $\in$. First, as in the proof of Lemma 8.4.3, we must introduce notation to reason about the tuples in the support of the distribution $p \in \mathit{prob}(v)$. The source state $(v, \mathbf{a})$, duration δ, flow vector $\mathbf{f}$, and symbolic state (v, η) of the transitions in question are implicit in the notation used below.

- We denote by $\mathsf{switch}[\tilde{p}_{\langle \mathbf{b} \rangle}](v', \mathbf{c}) = \{(v', \mathit{post}^i, X^i) \in \mathsf{support}(p) \mid \mathbf{c} = \mathbf{b}^i\}$ the set of tuples of $\mathsf{support}(p)$ which, when applied to the state $(v, \mathbf{a} + \delta.\mathbf{f})$, result in the state $(v', \mathbf{c})$, given that the vector of valuations $\langle \mathbf{b} \rangle = \mathsf{Combinations}((\mathbf{a} + \delta.\mathbf{f}), \mathsf{extract}(p))$ has been chosen.

- We use
$$\begin{aligned} & \mathsf{switch}[\hat{p}](v', \eta') \\ = \; & \{(v', \mathit{post}, X) \in \mathsf{support}(p) \mid (v', \eta') = \mathsf{post}[(v, v', \mathit{post}, X, p)](v, \eta)\} \end{aligned}$$
to denote the set of tuples of $\mathsf{support}(p)$ corresponding to edges which, when applied to the symbolic state (v, η) using the operation post, result in the symbolic state (v', η').

- Finally, we let $\mathsf{intersect}(s, s') = \mathsf{switch}[\tilde{p}_{\langle \mathbf{b} \rangle}](s) \cap \mathsf{switch}[\hat{p}](s')$.

The rest of the proof consists of two steps:

1. showing that a state s in the support of $\tilde{p}_{\langle \mathbf{b} \rangle}$ is contained within a symbolic state s' in the support of $\hat{p}$ if the *same tuple of original distribution* p derives both s and s';

2. showing that there exists a weight function for $(\tilde{p}_{\langle \mathbf{b} \rangle}, \hat{p})$ with respect to the inclusion relation $\in$.

Step 1. Formally, we show that, if $\mathsf{intersect}((v', \mathbf{c}), (v', \eta')) \neq \emptyset$ for the state $(v', \mathbf{c}) \in \mathsf{support}(\tilde{p})$ and the symbolic state $(v', \eta') \in \mathsf{support}(\hat{p})$, then $(v', \mathbf{c}) \in (v', \eta')$. Assume that the set $\mathsf{intersect}((v', \mathbf{c}), (v', \eta'))$ is non-empty, and consider any tuple $(v', \mathit{post}, X) \in \mathsf{intersect}((v', \mathbf{c}), (v', \eta'))$. Then, the tuple (v', post, X) corresponds to an index $1 \leq j \leq |\mathsf{support}(p)|$. Given the vector $\langle \mathbf{b} \rangle \in \mathsf{Combinations}((\mathbf{a} + \delta.\mathbf{f}), \mathsf{extract}(p))$, the valuation $\mathbf{c}$ is equal to the jth valuation $\mathbf{b}^j$ in the vector $\langle \mathbf{b} \rangle$. As we must have $\mathbf{b}^j \in (\mathbf{a} + \delta.\mathbf{f})[X := \mathit{post}]$, we also have $\mathbf{c} \in (\mathbf{a} + \delta.\mathbf{f})[X := \mathit{post}]$.

By the definitions of post and the forward reachability graph $\mathcal{S}_R^{\mathit{Target}}$ (Definition 9.2.3 and Definition 9.3.1, respectively), we have $(v', \eta') = \mathsf{post}[(v, v', \mathit{post}, X, p)](v, \eta)$. Then our aim can be restated as showing that $(v', \mathbf{c}) \in \mathsf{post}[(v, v', post, X, p)](v, \eta)$. From the fact that, for any symbolic state (v'', η''), we have $\mathsf{time\_succ}(v'', \eta'') \supseteq (v'', \eta'')$, then

$$\begin{aligned} \mathsf{post}[(v, v', post, X, p)](v, \eta) &= \mathsf{time\_succ}(\mathsf{disc\_succ}((v, v', post, X, p), (v, \eta)) \\ &\supseteq \mathsf{disc\_succ}((v, v', post, X, p), (v, \eta)) \ . \end{aligned}$$

Then, to show that $(v', \mathbf{c}) \in \mathsf{post}[(v, v', post, X, p)](v, \eta)$, it suffices to prove that

$$(v', \mathbf{c}) \in \mathsf{disc\_succ}((v, v', post, X, p), (v, \eta)) \ .$$

As the location component of both $(v', \mathbf{c})$ and $\mathsf{disc\_succ}((v, v', post, X, p), (v, \eta))$ is v', it remains to show that the valuation $\mathbf{c}$ of the state $(v', \mathbf{c})$ is included in the polyhedron of the symbolic state $\mathsf{disc\_succ}((v, v', post, X, p), (v, \eta))$. Therefore, by the definition of $\mathsf{disc\_succ}$ (Definition 9.2.3), our problem can be restated as that of showing $\mathbf{c} \in ((\eta \cap pre_v(p))[X := post]) \cap inv(v')$. As we have established that $\mathbf{c} \in (\mathbf{a} + \delta.\mathbf{f})[X := post]$, it suffices to show that $(\mathbf{a} + \delta.\mathbf{f})[X := post] \subseteq ((\eta \cap pre_v(p))[X := post]) \cap inv(v')$.

Consider any variable $x_i \in \mathcal{X}$, and let $\langle\!\langle l, u \rangle\!\rangle_i = post_i$ (recall that we assume that *post* is a tightened post condition), where $\langle\!\langle \in \{(, [\}$ and $\rangle\!\rangle \in \{),]\}$. There are two cases, depending on whether $x_i \in X$.

Case: $x_i \in X$. Then it follows from the definition of the reset operation applied to valuations and polyhedra that $(\mathbf{a} + \delta.\mathbf{f})[X := post]_i = \langle\!\langle l, u \rangle\!\rangle_i$ and, by the assumption of tightened post conditions (Definition 7.2.2), we have $((\eta \cap pre_v(p))[X := post])_i = \langle\!\langle l, u \rangle\!\rangle_i$. Hence $(\mathbf{a} + \delta.\mathbf{f})[X := post]_i = ((\eta \cap pre_v(p))[X := post])_i$.

Case: $x_i \notin X$. Then the reset operation results in $(\mathbf{a} + \delta.\mathbf{f})[X := post]_i = (\mathbf{a} + \delta.\mathbf{f})_i$, and $((\eta \cap pre_v(p))[X := post])_i = (\eta \cap pre_v(p))_i$. From $\mathbf{a} + \delta.\mathbf{f} \in \eta$ and $\mathbf{a} + \delta.\mathbf{f} \in pre_v(p)$, both of which we established in the *enabledness* part of this lemma, it then follows that $(\mathbf{a} + \delta.\mathbf{f})_i \in (\eta \cap pre_v(p))_i$.

Repeating this for all variables $x_i \in \mathcal{X}$ gives us that

$$(\mathbf{a} + \delta.\mathbf{f})[X := post] \subseteq ((\eta \cap pre_v(p))[X := post]) \ .$$

As $(\mathbf{a} + \delta.\mathbf{f})[X := post] \subseteq inv(v')$, because we assume that R has tightened post and pre-conditions (Definition 7.2.2 and Definition 7.2.3, respectively), then

$$(\mathbf{a} + \delta.\mathbf{f})[X := post] \subseteq ((\eta \cap pre_v(p))[X := post]) \cap inv(v') \ .$$

This is the result that we required to show that $(v', \mathbf{c}) \in (v', \eta')$.

Step 2. We now use Step 1 to obtain a weight function for the distributions $(\tilde{p}_{\langle \mathsf{b} \rangle}, \hat{p})$ with respect to $\in$. In a similar manner to the weight function featured in the proof of Lemma 8.4.6, we define the function $w : S_R \times \langle S \rangle \to [0, 1]$ by

$$w(s, s') = \sum_{q \in \mathsf{intersect}(s, s')} p(q)$$

for the state $s \in S_R$ and the symbolic state $s' \in \langle S \rangle$. We now show that w is an appropriate weight function (see Definition 3.2.12). Our first task is to prove that $w(s,s') > 0$ implies $s \in s'$, where $s \in S_R$ is a state of $\mathcal{S}_R$ and $s' \in \langle S \rangle$ is a symbolic state of $\mathcal{S}_R^{\mathit{Target}}$. By the definition of w, we have $w(s,s') > 0$ if and only if there exists a tuple $q \in \mathsf{intersect}(s,s')$. Furthermore, Step 1 establishes that the existence of $q \in \mathsf{intersect}(s,s')$ implies $s \in s'$. Then $w(s,s') > 0$ implies $s \in s'$. Hence, we have satisfied the first condition of a weight function.

The second task is to show that, for a given state $s \in S_R$, we have

$$\sum_{s' \in \langle S \rangle} w(s,s') = \tilde{p}_{\langle \mathbf{b} \rangle}(s) \; .$$

The following proof sketch follows a similar argument to that used in Lemma 8.4.6 for a similar result. Using the problem reduction steps used in the proof of Lemma 8.4.6, we can reduce our problem to that of showing

$$\sum_{s'' \in \mathsf{support}(\hat{p})} \; \sum_{q \in \mathsf{intersect}(s,s'')} p(q) = \sum_{q \in \mathsf{switch}[\tilde{p}_{\langle \mathbf{b} \rangle}](s)} p(q) \; .$$

Now, by Lemma 3.1, this problem can in turn be reduced to showing that the set of sets $\{\mathsf{intersect}(s,s'') \mid s'' \in \mathsf{support}(\hat{p})\}$ is an $\emptyset$-inclusive partition of $\mathsf{switch}[\tilde{p}_{\langle \mathbf{b} \rangle}](s)$. We can verify this using a similar technique to that used in the relevant part of the proof of Lemma 8.4.6.

Finally, for a given symbolic state $s' \in \langle S \rangle$, we can show that

$$\sum_{s \in S_R} w(s,s') = \hat{p}(s')$$

using symmetric arguments to those used above. Therefore, w is a weight function for $(\tilde{p}_{\langle \mathbf{b} \rangle}, \hat{p})$ with respect to $\in$, which concludes the proof of Lemma 9.4.1. $\square$

9.4.2 Properties of adversaries

In this section, we describe the way in which, for any *deterministic* adversary A of the concurrent probabilistic system $\mathcal{S}_R$ of R, a *randomised* adversary B of the forward reachability graph $\mathcal{S}_R^{\mathit{Target}}$ can be constructed such that the probability of reaching the target set *Target* is the same for A and B. This is similar to the adversary construction process for the reachability problem of probabilistic timed automata in Section 6.6.2, except in that case a *deterministic* adversary of the forward reachability graph could be constructed which reached the target set with the same probability as the concurrent probabilistic system adversary.

As in Section 6.6.2, our approach is to start from the initial state $(\bar{v}, \mathit{init}(\bar{v}))$ of $\mathcal{S}_R$ and progress along the length of paths of A. For paths of A of length $i \in \mathbb{N}$, by induction we will have constructed paths of the adversary B also of length i; we then define a choice of transitions of B after each path of length i so that these choices *emulate* the transitions chosen by A after paths of this length. More precisely, this emulation will comprise a randomised choice over event-distribution pairs, where each such pair is chosen on the basis of an event-distribution pair chosen by A after some path of length i; these two pairs must agree on their event component, and their distributions must be related by a weight function with respect to the inclusion relation $\in$. That this emulation is possible follows from Lemma 9.4.1 and the fact that we construct B in such a way as to ensure that every state reached by A after the ith transition is included within a symbolic state reached by B after the same number of transitions.

Observe that, for a certain length of paths $i \in \mathbb{N}$, we will have a *partially constructed* version of B; more precisely, we will have information on the choices of B for paths of length i or less. However, to reason about the properties of paths of length i, both of the adversary A and the adversary B, it is convenient to regard this partial construction as defining an adversary on paths of *any* finite length. Therefore, we introduce an infinite family of randomised adversaries $\langle B_i \rangle_{i \in \mathbb{N}}$ which is defined in the following manner: for $i = 0$, the adversary B_0 makes a choice for the path consisting of the initial symbolic state $\langle \bar{s} \rangle$ according to the choice of A from the initial state $(\bar{v}, \mathit{init}(\bar{v}))$ of $\mathcal{S}_R$, and can be arbitrary for any other path; for all $i \geq 1$, the adversary B_i makes a choice for all paths of length i identified by the adversary B_{i-1} according to the choices of A from paths of length i, and acts like B_{i-1} for all other paths. This is sufficient for our purposes, as, for any $i \in \mathbb{N}$, we consider properties of paths of length $i + 1$ only (observe that the adversary B_i defines paths of length $i + 1$). As i approaches the limit, the adversary B_i will define B.

An important exception to this construction process concerns paths of A and B that have already reached the target set *Target*. In this case, the only behaviour that can be taken in such a path of B is a self-loop, regardless of the behaviour of the adversary A in paths that have reached *Target*. This restriction has no effect on the result which we intend to prove concerning the probabilities of reaching the target set.

The proof of the required properties of adversaries is more involved than that featured in Section 6.6.2, as the relationship on transitions of states and symbolic states is less restricted than that featured in the probabilistic timed automata case, as the variable resetting mechanism is more liberal in the rectangular context. Indeed, the characterisation of the relationship between distributions of $\mathcal{S}_R$ and $\mathcal{S}_R^{\mathit{Target}}$ in terms of weight functions, and the use of randomised adversaries, means that the proof presented below has much in common with result concerning simulation and $\forall$PBTL properties presented in Appendix A.

Preliminaries

We first introduce two methods for reasoning about the properties of the finite paths of adversaries such as A and B. The initial task is to define an *inclusion-based path relation* which can be used to associate paths of A and B. Our aim is that, for a finite path π of B which is in the relation with a path ω of A, we can use the choice of event-distribution pair of A after ω to identify a possible choice of B after π.

Intuitively, the definition of the inclusion-based path relation requires that related paths are of the same length, and that the jth state along a path ω must be contained within the jth symbolic state along π, but only up to and including the first time that a location in the set *Target* is reached (naturally, the requirement on the inclusion of states in symbolic states means that related paths will reach this set at the same point). We use this fact in the adversary construction process in the next section, as it allows us to define B as making self-loop transitions in a location if the set *Target* has been reached; this suffices to show the correctness of our verification approach.

As in Section 6.6.2, we define the length of a finite path $\omega \in \mathit{Path}_{\mathit{fin}}^{\mathcal{S}_R}$ of the form

$$\omega = (v_0, \mathbf{a}_0) \xrightarrow{\delta_0} \xrightarrow{\theta_{p_0}, \tilde{p}_0} (v_1, \mathbf{a}_1) \xrightarrow{\delta_1} \xrightarrow{\theta_{p_1}, \tilde{p}_1} \ldots \xrightarrow{\delta_{n-1}} \xrightarrow{\theta_{p_{n-1}}, \hat{p}_{n-1}} (v_n, \mathbf{a}_n) \ ,$$

to be n, rather that $2n$. Similarly, when referring to the jth state along the path ω, then by $\omega(j)$ for $j \leq k$, we refer to the state $(v_j, \mathbf{a}_j)$.

Definition 9.4.2 *Let R be a probabilistic rectangular automaton with the associated concurrent probabilistic system $\mathcal{S}_R$, let $\mathit{Target} \subseteq V$ be a target set, and let $\mathcal{S}_R^{\mathit{Target}}$ be the resulting forward reachability graph. The* inclusion-based path relation $\in^{\mathit{Target}} \subseteq \mathit{Path}_{\mathit{fin}}^{\mathcal{S}_R} \times \mathit{Path}_{\mathit{fin}}^{\mathcal{S}_R^{\mathit{Target}}}$ *is such that, for every finite path $\omega \in \mathit{Path}_{\mathit{fin}}^{\mathcal{S}_R}, \pi \in \mathit{Path}_{\mathit{fin}}^{\mathcal{S}_R^{\mathit{Target}}}$, we have $\omega \in^{\mathit{Target}} \pi$ if and only if:*

1. $|\omega| = |\pi|$,
2. *$\omega(j) \in \pi(j)$ for all $j \leq i$, where $i = \min\{|\omega|, k \leq |\omega| \mid \mathsf{discrete}(\omega(k)) \in \mathit{Target}\}$.*

Secondly, we define the two countably infinite families of *path distributions* $\langle \mathbf{p}_i^A \rangle_{i \geq 1}$, $\langle \mathbf{p}_i^B \rangle_{i \geq 1}$, where, for each $i \geq 1$, the distribution $\mathbf{p}_i^A$ is derived from the probabilities assigned by $\mathit{Prob}_{\mathit{fin}}^A$ to the finite paths of A of length i such that their first state is $(\bar{v}, \mathit{init}(\bar{v}))$. Similarly, for each $i \geq 1$, the distribution $\mathbf{p}_i^B$ is derived from the probabilities assigned by $\mathit{Prob}_{\mathit{fin}}^{B_{i-1}}$ to the finite paths of B_{i-1} of length i such that their first state is $\langle \bar{s} \rangle$ (again, it is useful to recall that the adversary B_{i-1} defines paths of length i) [1].

[1] This path length is sufficient for the distribution $\mathbf{p}_i^B$ to reflect accurately the probability that the "limit adversary" B assigns to paths of length i. We chose to introduce the the path distribution $\mathbf{p}_i^B$ as relating to the adversary B_{i-1} as this is the adversary with the minimal index for which the induced path distribution agrees with the path distribution corresponding to the "limit adversary" B.

Definition 9.4.3 (Path distributions) *For any (randomised) adversary A of the concurrent probabilistic system $\mathcal{S}$, for any state $s \in S$, and for any $i \geq 1$, the* path distribution $\mathbf{p}^A_{i,s} \in \mu(Path^A_i(s))$ *is the distribution defined in the following way. For every finite path $\omega \in Path^A_i(s)$, we have:*

$$\mathbf{p}^A_{i,s}(\omega) \stackrel{\text{def}}{=} Prob^A_{fin}(\omega) \ .$$

It is straightforward to verify that path distributions are indeed distributions. Also note that we write $\mathbf{p}^A_i$ for $\mathbf{p}^A_{i,(\bar{v},init(\bar{v}))}$ and $\mathbf{p}^B_i$ for $\mathbf{p}^B_{i,\langle s \rangle}$ for simplicity.

Our aim is then to show that, for each $i \geq 1$, there exists a weight function $\mathbf{w}_i$ for $(\mathbf{p}^A_i, \mathbf{p}^B_i)$ with respect to $\in^{Target}$. The existence of $\mathbf{w}_i$ then permits us to define the next choices of B (that is, the choices of the adversary B_i) *and* provides us with the appropriate mechanism to reason about the probability of the reachability of the target set *Target* for the paths of A and B from $(\bar{v}, init(\bar{v}))$ and $\langle \bar{s} \rangle$ respectively, as explained in the proof of Proposition 9.4.5 later in this section.

Adversary construction process

We now describe the adversary construction process. For simplicity, we abbreviate $(\bar{v}, init(\bar{v}))$ as $\bar{s}$.

Proposition 9.4.4 (Adversary construction (rectangular)) *Consider the probabilistic rectangular automaton R with the associated concurrent probabilistic system $\mathcal{S}_R$, the target set $Target \subseteq V$ of locations, and the resulting forward reachability graph $\mathcal{S}^{Target}_R$. Let A be a deterministic adversary of $\mathcal{S}_R$. Then there exists a countable infinite family of randomised adversaries $\langle B_i \rangle_{i \in \mathbb{N}}$ of $\mathcal{S}^{Target}_R$ such that $\mathbf{p}^A_i \in^{Target} \mathbf{p}^B_i$ for each $i \geq 1$.*

Proof. We proceed by induction on the length of paths in $Path^A_{fin}(\bar{s})$, which coincides with induction on the index of the adversaries in the family $\langle B_i \rangle_{i \in \mathbb{N}}$.

Base. We determine the randomised adversary B_0 of the forward reachability graph $\mathcal{S}^{Target}_R$ from the paths of length 0 of A; that is, the path consisting of the single-state $\bar{s} = (\bar{v}, init(\bar{v}))$. Then we have two cases, depending on whether $\bar{v} \in Target$.

Case $\bar{v} \in Target$**.** Clearly, $\mathsf{discrete}(\langle \bar{s} \rangle) \in Target$ also. We let the choice of B_0 be arbitrary for *all* paths. To see why this results in the existence of a weight function for $\mathbf{p}^A_1, \mathbf{p}^B_1$ with respect to $\in^{Target}$, consider the following argument. First note that, for *all* paths $\omega \in Path^A_1(\bar{s}), \pi \in Path^{B_0}_1(\langle \bar{s} \rangle)$, we have $\omega \in^{Target} \pi$. To see this, note that $|\omega| = |\pi| = 1$, thus satisfying condition (1) of the definition of $\in^{Target}$ (Definition 9.4.2). For condition (2), we note that $\min\{|\omega|, k \leq |\omega| \mid \mathsf{discrete}(\omega(k)) \in Target\} = 0$; then the condition reduces to requiring that $\omega(0) \in \pi(0)$, which is the same as $\bar{s} \in \langle s \rangle$. Therefore, we have $\omega \in^{Target} \pi$ for all paths

$\omega \in Path_1^A(\bar{s}), \pi \in Path_1^{B_0}(\langle \bar{s} \rangle)$. Then, whatever the form of B_0 (and therefore whatever the form of paths in the set $Path_1^{B_0}(\langle \bar{s} \rangle))$, we have $\mathbf{p}_1^A \in^{Target} \mathbf{p}_1^B$. This is because it is trivial to find a weight function for $\mathbf{p}_1^A, \mathbf{p}_1^B$ with respect to $\in^{Target}$ if all the elements in the support of these distributions are related by $\in^{Target}$.

Case $\bar{v} \notin Target$**.** Let $\bar{s} \xrightarrow{\delta} \xrightarrow{\theta_p, \tilde{p}}$ be the transition taken by A in the state $\bar{s}$ (we omit the vector subscript on $\tilde{p}$ for clarity). That is, the adversary A is such that $A(\bar{s})(\delta, \mathcal{D}(\bar{s} + \delta)) = 1$ and $A(\bar{s} + \delta)(\theta_p, \tilde{p}) = 1$. Then, from $\bar{s} \in \langle \bar{s} \rangle$ and Lemma 9.4.1, there exists a transition $\langle \bar{s} \rangle \xrightarrow{\theta_p, \hat{p}}$ such that $\tilde{p} \in \hat{p}$; that is, there exists the event-distribution pair $(\theta_p, \hat{p}) \in Steps(\langle \bar{s} \rangle)$. Then we let $B_0(\langle \bar{s} \rangle)(\theta_p, \hat{p}) = 1$, and B_0 can be arbitrary for all other paths.

We now show that there exists a weight function $\mathbf{w}_1$ for $(\mathbf{p}_1^A, \mathbf{p}_1^B)$ with respect to $\in^{Target}$. We show that $\mathbf{w}_1$ can be obtained from the weight function w for $(\tilde{p}, \hat{p})$ with respect to $\in$, which exists by $\tilde{p} \in \hat{p}$. First, we identify the paths in $Path_1^A(\bar{s}), Path_1^{B_0}(\langle \bar{s} \rangle)$ which are related by $\in^{Target}$. Recall the definition of $\in^{Target}$ (Definition 9.4.2). Condition (1) is trivially satisfied by all paths in $Path_1^A(\bar{s}), Path_1^{B_0}(\langle \bar{s} \rangle)$. Now consider condition (2). As $\bar{s} \in \langle \bar{s} \rangle$, for any paths $\omega \in Path_1^A(\bar{s}), \pi \in Path_1^{B_0}(\langle \bar{s} \rangle)$, we have $\omega \in^{Target} \pi$ if and only if $\omega(1) \in \pi(1)$. Then, as w is a weight function for $(\tilde{p}, \hat{p})$ with respect to $\in$, for $\omega \in Path_1^A(\bar{s}), \pi \in Path_1^{B_0}(\langle \bar{s} \rangle)$ where $\omega = \bar{s} \xrightarrow{\delta} \xrightarrow{\theta_p, \tilde{p}} s$ and $\pi = \langle \bar{s} \rangle \xrightarrow{\theta_p, \hat{p}} s'$, let $\mathbf{w}_1(\omega, \pi) = w(s, s')$.

The verification that $\mathbf{w}_1$ is indeed a weight function for $(\mathbf{p}_1^A, \mathbf{p}_1^B)$ with respect to $\in^{Target}$ proceeds in the following manner. First we require that $\mathbf{w}_1(\omega, \pi) > 0$ implies that $\omega \in^{Target} \pi$, for all $\omega \in Path_1^A(\bar{s})$, $\pi \in Path_1^{B_0}(\langle \bar{s} \rangle)$. As $|\omega| = |\pi| = 1$, the first condition of $\in^{Target}$ (Definition 9.4.2) is satisfied trivially. Consider condition (2) of the definition of $\in^{Target}$. It follows from the definition of $\mathbf{w}_1(\omega, \pi) > 0$ implies $w(\omega(1), \pi(1)) > 0$, which implies $\omega(1) \in \pi(1)$. Then, as $\omega(0) \in \pi(0)$ (from $\bar{s} \in \langle \bar{s} \rangle$) and $\omega(1) \in \pi(1)$, we must have $\omega \in^{Target} \pi$.

Next, we must show that $\sum_{\pi \in Path_1^{B_0}(\langle \bar{s} \rangle)} \mathbf{w}_1(\omega, \pi) = \mathbf{p}_1^A(\omega)$ for all $\omega \in Path_1^A(\bar{s})$. Then:

$$\begin{aligned}
\sum_{\pi \in Path_1^{B_0}(\langle \bar{s} \rangle)} \mathbf{w}_1(\omega, \pi) &= \sum_{s' \in \mathsf{support}(\hat{p})} w(last(\omega), s') \\
&\qquad \text{(by definition of } \mathbf{w}_1 \text{ and } Path_1^{B_0}(\langle \bar{s} \rangle)) \\
&= \tilde{p}(last(\omega)) \\
&\qquad \text{(by Definition 3.2.12)} \\
&= \mathbf{p}_1^A(\omega) \\
&\qquad \text{(by Definition 3.2.2, Definition 9.4.3).}
\end{aligned}$$

Similarly (using the same manipulations and justifications), for any path $\pi \in$

$\mathit{Path}_1^{B_0}(\langle \bar{s} \rangle)$, we have:

$$\sum_{\omega \in \mathit{Path}_1^A(\bar{s})} \mathbf{w}_1(\omega, \pi) = \mathbf{p}_1^B(\pi) \ .$$

We conclude that $\mathbf{w}_1$ is a weight function for $(\mathbf{p}_1^A, \mathbf{p}_1^B)$ with respect to $\in^{\mathit{Target}}$.

Induction. Consider any $i \geq 1$. By induction, we assume the existence of a randomised adversary B_{i-1}, which is such that there exists a weight function $\mathbf{w}_i$ for $(\mathbf{p}_i^A, \mathbf{p}_i^B)$ with respect to $\in^{\mathit{Target}}$. We detail the construction of the randomised adversary B_i, and show that there exists a weight function $\mathbf{w}_{i+1}$ for $(\mathbf{p}_{i+1}^A, \mathbf{p}_{i+1}^B)$ with respect to $\in^{\mathit{Target}}$.

The distinction between the case in which paths of interest have reached a target location, and the case in which they have not, have to be treated in entirely different fashions. Therefore, we now split the proof into two cases, which are subsequently reconciled in the final section of the proof. The aim in each of the cases is to show the existence of a weight function which relates sets of paths of length $i+1$ which are identified by A and B_i. These weight functions take distinct forms in each of the two cases, but they can both be used to define the overall weight function $\mathbf{w}_{i+1}$ for $(\mathbf{p}_{i+1}^A, \mathbf{p}_{i+1}^B)$ with respect to $\in^{\mathit{Target}}$.

Firstly, for every path $\pi \in \mathit{Path}_i^{B_{i-1}}(\langle \bar{s} \rangle)$, we have the following two cases.

Case: there exists $j \leq i$ such that $\mathsf{discrete}(\pi(j)) \in \mathit{Target}$. Recall that, from the construction of the forward reachability graph $\mathcal{S}_R^{\mathit{Target}}$ (Definition 9.3.1), there is only *one* choice available to B_i in a symbolic state for which the location is in *Target*: that of selecting the self-loop back to the current symbolic state. Formally, if $\pi \in \mathit{Path}_i^{B_{i-1}}(\langle \bar{s} \rangle)$ is such that $\mathsf{discrete}(\pi(j)) \in \mathit{Target}$ for some $j \leq i$, then let $B_i(\pi)(\perp, \mathcal{D}(\pi(i))) = 1$ (note that $\mathit{Steps}(\pi(i)) = \{(\perp, \mathcal{D}(\pi(i)))\}$ by construction of $\mathcal{S}_R^{\mathit{Target}}$).

Next, we consider distributions of paths after a prefix of length i which are induced by the choices made by A and B_i. For any path $\omega \in \mathit{Path}_i^A(\bar{s})$, we let $\mathbf{p}_\omega \in \mu(\mathit{Path}_{i+1}^A(\bar{s}))$ be the distribution defined by

$$\mathbf{p}_\omega(\omega') = \begin{cases} \tilde{p}(s) & \text{if } \omega' = \omega \xrightarrow{\delta} \xrightarrow{\theta_p, \tilde{p}} s \text{ and } \omega' \in \mathit{Path}_i^A(\bar{s}) \\ 0 & \text{otherwise.} \end{cases}$$

Similarly, for the path under consideration, $\pi \in \mathit{Path}_i^{B_{i-1}}(\langle \bar{s} \rangle)$, we let $\mathbf{p}_\pi \in \mathit{Path}_{i+1}^{B_i}(\langle \bar{s} \rangle)$ be the distribution defined by

$$\mathbf{p}_\pi(\pi') = \begin{cases} 1 & \text{if } \pi' = \pi \xrightarrow{\perp, \mathcal{D}(s)} s \\ 0 & \text{otherwise.} \end{cases}$$

Note that $\mathbf{p}_\omega$ and $\mathbf{p}_\pi$ are *not* path distributions in the sense of Definition 9.4.3. Next, consider the function $\mathbf{w}_{\omega\pi} : \mathit{Path}_{i+1}^A(\bar{s}) \times \mathit{Path}_{i+1}^{B_i}(\langle \bar{s} \rangle) \to [0,1]$ defined by

$\mathbf{w}_{\omega\pi}(\omega',\pi') = \mathbf{p}_\omega(\omega')$. We now show that $\mathbf{w}_{\omega\pi}$ is a weight function for $(\mathbf{p}_\omega, \mathbf{p}_\pi)$ with respect to $\in^{\mathit{Target}}$ (see Definition 3.2.12).

First, we require that $\mathbf{w}_{\omega\pi}(\omega',\pi') > 0$ implies $\omega' \in^{\mathit{Target}} \pi'$ for any $\omega' \in \mathit{Path}^A_{i+1}(\bar{s})$, $\pi' \in \mathit{Path}^{B_i}_{i+1}(\langle\bar{s}\rangle)$. By definition, we have $|\omega'| = |\pi'| = i+1$, and, as there exists $j < i$ such that $\mathsf{discrete}(\pi'(j)) \in \mathit{Target}$, it must be the case that $\mathsf{discrete}(\omega'(j)) \in \mathit{Target}$; therefore, both of the conditions of the definition of $\in^{\mathit{Target}}$ (Definition 9.4.2) are satisfied, and hence $\mathbf{w}_{\omega\pi}(\omega',\pi') > 0$ implies $\omega' \in^{\mathit{Target}} \pi'$.

Secondly, we must show that $\sum_{\pi'' \in \mathit{Path}^{B_i}_{i+1}(\langle\bar{s}\rangle)} \mathbf{w}_{\omega\pi}(\omega',\pi') = \mathbf{p}_\omega(\omega')$ for each $\omega' \in \mathit{Path}^A_{i+1}(\bar{s})$. As there exists *one* particular path $\pi' \in \mathit{Path}^{B_i}_{i+1}(\langle\bar{s}\rangle)$ such that $\mathbf{w}_{\omega\pi}(\omega',\pi') > 0$, and that $\mathbf{w}_{\omega\pi}(\omega',\pi') = \mathbf{p}_\omega(\omega')$ for this path, then the result follows. Similarly, for the path $\pi' \in \mathit{Path}^{B_i}_{i+1}(\langle\bar{s}\rangle)$, we have $\mathbf{p}_\pi(\pi') = 1$, and, as $\sum_{\omega' \in \mathit{Path}^A_{i+1}(\bar{s})} \mathbf{p}_\omega(\omega') = 1$, we have

$$\sum_{\omega' \in \mathit{Path}^A_{i+1}(\bar{s})} \mathbf{w}_{\omega\pi}(\omega',\pi') = \sum_{\omega' \in \mathit{Path}^A_{i+1}(\bar{s})} \mathbf{p}_\omega(\omega') = 1$$

which is, of course, equal to $\mathbf{p}_\pi(\pi')$. Therefore, we also have

$$\sum_{\omega' \in \mathit{Path}^A_{i+1}(\bar{s})} \mathbf{w}_{\omega\pi}(\omega',\pi') = \mathbf{p}_\pi(\pi') \ ,$$

and all of the conditions for $\mathbf{w}_{\omega\pi}$ to be a weight function for $(\mathbf{p}^A_{i+1}, \mathbf{p}^B_{i+1})$ with respect to $\in^{\mathit{Target}}$ are satisfied.

Case: $\mathsf{discrete}(\pi(j)) \notin \mathit{Target}$ **for all** $j \leq i$**.** For the path $\omega \in \mathit{Path}^A_i(\bar{s})$, let

$$\mathsf{Random}_i(\omega,\pi) = \frac{\mathbf{w}_i(\omega,\pi)}{\mathbf{p}^B_i(\pi)} \ .$$

For each path $\omega \in \mathit{Path}^A_i(\bar{s})$ such that $\mathbf{w}_i(\omega,\pi) > 0$, we observe that $\omega \in^{\mathit{Target}} \pi$, which implies that there does not exist a $j \leq i$ such that $\mathsf{discrete}(\omega(j)) \in \mathit{Target}$ for the case of ω, also. Therefore, from the fact that we are considering the case in which there does not exist $j \leq i$ such that $\mathsf{discrete}(\omega(j)) \in \mathit{Target}$, and by condition (2) of the definition of $\in^{\mathit{Target}}$ (Definition 9.4.2), we must have $\omega(i) \in \pi(i)$. Then, by Lemma 9.4.1, we can match the transition $\omega(i) \xrightarrow{\delta} \xrightarrow{\theta_p,\tilde{p}}$ by the transition $\pi(i) \xrightarrow{\theta_p,\hat{p}}$ such that $\tilde{p} \in \hat{p}$. We select arbitrarily such an event-distribution pair $(\theta_p, \hat{p}) \in \mathit{Steps}(\pi(i))$, and denote it by $\mathsf{Choice}_i(\omega,\pi)$. Furthermore, we let

$$\mathit{Path}^A_{\theta_p,\hat{p}}(\bar{s}) = \{\omega \in \mathit{Path}^A_i(\bar{s}) \mid \mathsf{Choice}_i(\omega,\pi) = (\theta_p,\hat{p})\} \ .$$

Finally, we let

$$B_i(\pi)(\theta_p,\hat{p}) = \sum_{\omega \in \mathit{Path}^A_{\theta_p,\hat{p}}(\bar{s})} \mathsf{Random}_i(\omega,\pi) \ .$$

The fact that $B_i(\pi)$ is a distribution over the set $\mathit{Steps}(\pi(i))$ follows in the same manner as the similar result in the proof of Proposition A.2.2 in Appendix A.

As in the preceding case, we now express the distributions over states chosen after paths of length i as distributions over *paths*. For the path $\omega \in \mathit{Path}_i^A(\bar{s})$, let $\mathbf{p}_\omega \in \mu(\mathit{Path}_{i+1}^A(\bar{s}))$ be the distribution defined in exactly the same manner as in the preceding case. For the paths $\omega \in \mathit{Path}_i^A(\bar{s})$, $\pi \in \mathit{Path}_i^{B_{i-1}}(\langle \bar{s} \rangle)$ which are such that $\mathbf{w}_i(\omega, \pi) > 0$, let $\mathbf{p}_{\omega\pi} \in \mu(\mathit{Path}_{i+1}^{B_i}(\langle \bar{s} \rangle))$ be the distribution over paths defined as follows. For the event-distribution pair $(\theta_p, \hat{p}) \in \mathit{Steps}(\pi(i))$ which is such that $(\theta_p, \hat{p}) = \mathsf{Choice}_i(\omega, \pi)$, we let

$$\mathbf{p}_{\omega\pi}(\pi') = \begin{cases} \hat{p}(s) & \text{if } \pi' = \pi \xrightarrow{\theta_p, \hat{p}} s \\ 0 & \text{otherwise.} \end{cases}$$

Say the transition induced by A after the path ω is $\omega \xrightarrow{\delta} \xrightarrow{\theta_p, \tilde{p}}$. Recall that, by the definition of B_i, for the paths ω, π, the distribution on states $\tilde{p}$, and the distribution on symbolic states $\hat{p}$, we have $\tilde{p} \in \hat{p}$, which denotes the existence of a weight function w for $(\tilde{p}, \hat{p})$ with respect to $\in$. Then we define the function $\mathbf{w}_{\omega\pi} : \mathit{Path}_{i+1}^A(\bar{s}) \times \mathit{Path}_{i+1}^{B_i}(\langle \bar{s} \rangle) \to [0,1]$ in the following manner: for $\omega' = \omega \xrightarrow{\sigma, p} s \in \mathit{Path}_{i+1}^A(\bar{s})$, $\pi' = \pi \xrightarrow{\sigma, p'} s' \in \mathit{Path}_{i+1}^{B_i}(\langle \bar{s} \rangle)$, simply let $\mathbf{w}_{\omega\pi}(\omega', \pi') = w(s, s')$. Then $\mathbf{w}_{\omega\pi}$ is a weight function for $(\mathbf{p}_\omega, \mathbf{p}_{\omega\pi})$ with respect to $\in^{\mathit{Target}}$.

To see this, we first require that $\mathbf{w}_{\omega\pi}(\omega', \pi') > 0$ implies that $\omega' \in^{\mathit{Target}} \pi'$ for the paths ω', π' described above. We show that ω' and π' satisfy the two conditions of $\in^{\mathit{Target}}$ (Definition 9.4.2). Condition (1) is satisfied, because we have $|\omega'| = |\pi'| = i+1$. For condition (2), observe that $\mathbf{w}_{\omega\pi}(\omega', \pi') > 0$ if and only if $w(\omega'(i+1), \pi'(i+1)) > 0$. Also, $w(\omega'(i+1), \pi'(i+1)) > 0$ implies that $\omega'(i+1) \in \pi'(i+1)$. This satisfies condition (2) of the definition of $\in^{\mathit{Target}}$, and therefore $\mathbf{w}_{\omega\pi}(\omega', \pi') > 0$ implies $\omega' \in^{\mathit{Target}} \pi'$.

Using this framework, the proofs of

$$\sum_{\pi' \in \mathit{Path}_{i+1}^{B_i}(\langle \bar{s} \rangle)} \mathbf{w}_{\omega\pi}(\omega', \pi') \;=\; \mathbf{p}_\omega(\omega') \,, \qquad \sum_{\omega' \in \mathit{Path}_{i+1}^A(\bar{s})} \mathbf{w}_{\omega\pi}(\omega', \pi') \;=\; \mathbf{p}_{\omega\pi}(\pi')$$

follow in a similar manner to the analogous results given in the proof of Proposition A.2.2. We conclude that the function $\mathbf{w}_{\omega\pi}$ is a weight function for $(\mathbf{p}_\omega, \mathbf{p}_{\omega\pi})$ with respect to $\in^{\mathit{Target}}$.

Repeating this process for all paths $\pi \in \mathit{Path}_i^{B_{i-1}}(\langle \bar{s} \rangle)$ results in a set of distributions over event-distribution pairs, each denoted by $B_i(\pi)$. Furthermore, for paths of length less than i, we let the randomised choices of B_i equal the randomised choices of B_{i-1} for these paths; that is, for each path $\pi \in \bigcup_{1 \leq j < i} \mathit{Path}_j^{B_{j-1}}(\langle \bar{s} \rangle)$, we let

$B_i(\pi) = B_{i-1}(\pi)$. The choices of B_i for all other paths is arbitrary. We have concluded our definition of the randomised adversary B_i.

We now show that B_i is such that the induction hypothesis is preserved; that is, $\mathbf{p}^A_{i+1} \in^{\mathit{Target}} \mathbf{p}^B_{i+1}$. Recall that $\mathbf{p}^A_{i+1} \in^{\mathit{Target}} \mathbf{p}^B_{i+1}$ if and only if there exists a weight function $\mathbf{w}_{i+1}$ for $(\mathbf{p}^A_{i+1}, \mathbf{p}^B_{i+1})$ with respect to $\in^{\mathit{Target}}$. We proceed to define the function $\mathbf{w}_{i+1} : \mathit{Path}^A_{i+1}(\bar{s}) \times \mathit{Path}^{B_i}_{i+1}(\langle\bar{s}\rangle) \to [0,1]$. For the path $\omega' \in \mathit{Path}^A_{i+1}(\bar{s})$, $\pi' \in \mathit{Path}^{B_i}_{i+1}(\langle\bar{s}\rangle)$, for which the ith prefix are denoted by $(\omega')^{(i)} = \omega$ and $(\pi')^{(i)} = \pi$, we let

$$\mathbf{w}_{i+1}(\omega', \pi') = \mathbf{w}_i(\omega, \pi).\mathbf{w}_{\omega\pi}(\omega', \pi') \ .$$

We only sketch the proof which shows that $\mathbf{w}_{i+1}$ is a weight function for $(\mathbf{p}^A_{i+1}, \mathbf{p}^B_{i+1})$ with respect to $\in^{\mathit{Target}}$ (the steps are similar to those used in the proof of Proposition A.2.2).

First, we show that $\mathbf{w}_{i+1}(\omega', \pi') > 0$ implies $\omega' \in^{\mathit{Target}} \pi'$ for paths $\omega' \in \mathit{Path}^A_{i+1}(\bar{s})$, $\pi' \in \mathit{Path}^{B_i}_{i+1}(\langle\bar{s}\rangle)$. It follows by definition that $|\omega'| = |\pi'| = i+1$, therefore satisfying the first condition for $\in^{\mathit{Target}}$ (Definition 9.4.2). Now consider the second condition. By the definition of $\mathbf{w}_{i+1}$, if $\mathbf{w}_{i+1}(\omega', \pi') > 0$, then $\mathbf{w}_i(\omega, \pi) > 0$; we then split the proof into two cases depending on whether ω and π have already reached a location in *Target* or not. If ω and π have reached such a location, then clearly $\omega' \in^{\mathit{Target}} \pi'$. If these paths have not yet reached a location in *Target*, then we require that $\omega'(i+1) \in \pi'(i+1)$; however, by the definition of $\mathbf{w}_{i+1}$, we have that $\mathbf{w}_{i+1}(\omega', \pi') > 0$ implies $\mathbf{w}_{\omega\pi}(\omega', \pi') > 0$, in turn implying $w(\omega'(i+1), \pi'(i+1)) > 0$, which finally implies $\omega'(i+1) \in \pi'(i+1)$. Hence $\mathbf{w}_{i+1}(\omega', \pi') > 0$ implies $\omega' \in^{\mathit{Target}} \pi'$.

To show that, for every path $\omega' \in \mathit{Path}^A_{i+1}(\bar{s})$, we have

$$\sum_{\pi' \in \mathit{Path}^{B_i}_{i+1}(\langle\bar{s}\rangle)} \mathbf{w}_{i+1}(\omega', \pi') = \mathbf{p}^A_{i+1}(\omega') \ ,$$

we can use the methodology of the proof of Proposition A.2.2, merely substituting in our new definition of $\mathbf{w}_{\omega\pi}$ (which can reflect both of the two cases featured throughout this proof). To show that, for every path $\pi' \in \mathit{Path}^{B_i}_{i+1}(\bar{s})$, we have

$$\sum_{\omega' \in \mathit{Path}^A_{i+1}(\bar{s})} \mathbf{w}_{i+1}(\omega', \pi') = \mathbf{p}^B_{i+1}(\pi')$$

requires a slightly more involved proof, as, yet again, we proceed by considering two cases. That is, if π has already reached a location in *Target*, we follow the argument of the proof of Proposition A.2.2 with the distribution $\mathbf{p}_\pi$ in place of $\mathbf{p}_{\omega\pi}$; otherwise, we proceed in exactly the same manner of the proof of Proposition A.2.2. We then conclude that $\mathbf{w}_{i+1}$ is a weight function for $(\mathbf{p}^A_{i+1}, \mathbf{p}^B_{i+1})$ with respect to $\in^{\mathit{Target}}$, which also concludes the proof of Proposition 9.4.4. □

Properties of the constructed adversaries

Our final task is to show that both the deterministic adversary A and the randomised adversary B reach the target set *Target* with the same probability.

Proposition 9.4.5 *Consider the probabilistic rectangular automaton R with the associated concurrent probabilistic system $\mathcal{S}_R$, the target set $Target \subseteq V$, and the resulting forward reachability graph $\mathcal{S}_R^{Target}$. For a deterministic adversary A of $\mathcal{S}_R$, let B be a randomised adversary of $\mathcal{S}_R^{Target}$ constructed from A using the technique of the proof of Proposition 9.4.4. Then, we have*

$$\begin{aligned} & Prob^A(\{\omega \in Path^A_{ful}(\bar{s}) \mid \exists i \in \mathbb{N}.\mathsf{discrete}(\omega(i)) \in Target\}) \\ = \quad & Prob^B(\{\pi \in Path^B_{ful}(\langle \bar{s} \rangle) \mid \exists i \in \mathbb{N}.\mathsf{discrete}(\pi(i)) \in Target\}) \; . \end{aligned}$$

Proof. We proceed inductively along the length of the paths of PMC^A and PMC^B, showing that, for each $i \in \mathbb{N}$, the probability of paths of length i that have reached a location in *Target* is the same for both PMC^A and PMC^B. Formally, for each $i \in \mathbb{N}$, we have

$$\begin{aligned} & Prob^A(\{\omega \in Path^A_{ful}(\bar{s}) \mid \exists j \leq i.\mathsf{discrete}(\omega(i)) \in Target\}) \\ = \quad & Prob^B(\{\pi \in Path^B_{ful}(\langle \bar{s} \rangle) \mid \exists j \leq i.\mathsf{discrete}(\pi(i)) \in Target\}) \; . \end{aligned}$$

Base. Consider the initial states $\bar{s}$, $\langle \bar{s} \rangle$ of all of PMC^A, PMC^B respectively. From the definitions of $\bar{s}$ and $\langle \bar{s} \rangle$, it must be the case that $\mathsf{discrete}(\bar{s}) = \mathsf{discrete}(\langle \bar{s} \rangle) = \bar{v}$, and therefore $\mathsf{discrete}(\bar{s}) \in Target$ if and only if $\mathsf{discrete}(\langle \bar{s} \rangle) \in Target$. Therefore

$$\begin{aligned} & Prob^A(\{\omega \in Path^A_{ful}(\bar{s}) \mid \mathsf{discrete}(\omega(0)) \in Target\}) \\ = \quad & Prob^B(\{\pi \in Path^B_{ful}(\langle \bar{s} \rangle) \mid \mathsf{discrete}(\omega(0)) \in Target\}) \; . \end{aligned}$$

Induction. Consider $i \geq 1$. Recall that we have defined the path distributions $\mathbf{p}^A_i$ and $\mathbf{p}^B_i$ such that, for any $\omega \in Path^A_i(\bar{s})$, we have $\mathbf{p}^A_i(\omega) = Prob^A_{fin}(\omega)$. Similarly, for any $\pi \in Path^B_i(\langle \bar{s} \rangle)$, we have $\mathbf{p}^B_i(\pi) = Prob^B_{fin}(\pi)$. Furthermore, we also have $\mathbf{p}^A_i \in^{Target} \mathbf{p}^B_i$ by Proposition 9.4.4.

Let $\Omega_{Target} \subseteq Path^A_i(\bar{s})$ be the set of paths of length i of A that have already reached a location in *Target*; formally,

$$\Omega_{Target} = \{\omega \in Path^A_i(\bar{s}) \mid \exists j \leq i.\mathsf{discrete}(\omega(j)) \in Target\} \; .$$

Similarly, let $\Pi_{Target} \subseteq Path^B_i(\langle \bar{s} \rangle)$ be the set of paths of length i of B that have reached a location in *Target*:

$$\Pi_{Target} = \{\pi \in Path^B_i(\langle \bar{s} \rangle) \mid \exists j \leq i.\mathsf{discrete}(\pi(j)) \in Target\} \; .$$

Consider the path relation $\in^{Target}$ (Definition 9.4.2). We show that any paths $\omega \in Path^A_i(\bar{s})$, $\pi \in Path^B_i(\langle \bar{s} \rangle)$ which are such that $\omega \in^{Target} \pi$ are also such that $\omega \in \Omega_{Target}$

if and only if $\pi \in \Pi_{Target}$. Observe the following facts given by the relation $\in^{Target}$. If there exists $j \leq i$ such that $\mathsf{discrete}(\omega(j)) \in Target$, then, from $\omega \in^{Target} \pi$, it must also be the case that $\mathsf{discrete}(\pi(j)) \in Target$. Hence, $\omega \in \Omega_{Target}$ and $\pi \in \Pi_{Target}$. Similarly, if $\mathsf{discrete}(\omega(j)) \notin Target$ for all $j \leq i$, then $\omega \in^{Target} \pi$ implies that $\omega(j) \in \pi(j)$ for all $j \leq i$. Therefore $\mathsf{discrete}(\pi(j)) \notin Target$ also. Hence, if $\omega \notin \Omega_{Target}$ then $\pi \notin \Pi_{Target}$. Thus, $\omega \in \Omega_{Target}$ if and only if $\pi \in \Pi_{Target}$. Then, from this fact and Lemma 3.2.14, we have

$$\sum_{\omega \in \Omega_{Target}} \mathbf{p}_i^A(\omega) = \sum_{\pi \in \Pi_{Target}} \mathbf{p}_i^B(\pi) \; . \tag{9.1}$$

Now, we have

$$\begin{aligned} \bigcup_{\omega \in \Omega_{Target}} \omega \uparrow &= \{\omega' \in Path^A_{ful}(\bar{s}) \mid \exists j \leq i.\mathsf{discrete}(\omega'(i)) \in Target\} \\ \bigcup_{\pi \in \Pi_{Target}} \pi \uparrow &= \{\pi' \in Path^B_{ful}(\langle \bar{s} \rangle) \mid \exists j \leq i.\mathsf{discrete}(\pi'(i)) \in Target\} \; . \end{aligned}$$

Then, from Lemma 3.2.4 and the definition of path distributions (Definition 9.4.3), we have

$$\begin{aligned} Prob^A(\{\omega \in Path^A_{ful}(\bar{s}) \mid \exists j \leq i.\mathsf{discrete}(\omega(j)) \in Target\}) &= \sum_{\omega \in \Omega_{Target}} \mathbf{p}_i^A(\omega) \\ Prob^B(\{\pi \in Path^B_{ful}(\langle \bar{s} \rangle) \mid \exists j \leq i.\mathsf{discrete}(\pi(j)) \in Target\}) &= \sum_{\pi \in \Pi_{Target}} \mathbf{p}_i^B(\pi) \; . \end{aligned}$$

Therefore, by Equation 9.1, we have:

$$\begin{aligned} & Prob^A(\{\omega \in Path^A_{ful}(\bar{s}) \mid \exists j \leq i.\mathsf{discrete}(\omega(j)) \in Target\}) \\ = \; & Prob^B(\{\pi \in Path^B_{ful}(\langle \bar{s} \rangle) \mid \exists j \leq i.\mathsf{discrete}(\pi(j)) \in Target\}) \; . \end{aligned}$$

As this equality on probability measures holds for all $i \in \mathbb{N}$, and observing that

$$\begin{aligned} & \{\omega \in Path^A_{ful}(\bar{s}) \mid \exists i \in \mathbb{N}.\mathsf{discrete}(\omega(i)) \in Target\} \\ = \; & \textstyle\bigcup_{i \in \mathbb{N}} \{\omega \in Path^A_{ful}(\bar{s}) \mid \exists j \leq i.\mathsf{discrete}(\omega(j)) \in Target\} \end{aligned}$$

and

$$\begin{aligned} & \{\pi \in Path^B_{ful}(\langle \bar{s} \rangle) \mid \exists i \in \mathbb{N}.\mathsf{discrete}(\pi(i)) \in Target\} \\ = \; & \textstyle\bigcup_{i \in \mathbb{N}} \{\pi \in Path^B_{ful}(\langle \bar{s} \rangle) \mid \exists j \leq i.\mathsf{discrete}(\pi(j)) \in Target\} \; , \end{aligned}$$

we conclude that

$$\begin{aligned} & Prob^A(\{\omega \in Path^A_{ful}(\bar{s}) \mid \exists i \in \mathbb{N}.\mathsf{discrete}(\omega(i)) \in Target\}) \\ = \; & Prob^B(\{\pi \in Path^B_{ful}(\langle \bar{s} \rangle) \mid \exists i \in \mathbb{N}.\mathsf{discrete}(\pi(i)) \in Target\}) \; . \end{aligned} \qquad \square$$

It follows from Proposition 9.4.4 and Proposition 9.4.5 that, for any deterministic adversary A of the concurrent probabilistic system $\mathcal{S}_R$ of the probabilistic rectangular automaton R, and the target set of locations *Target*, we can construct a randomised adversary B of the forward reachability graph $\mathcal{S}_R^{\mathit{Target}}$ such that the probability of reaching a location in *Target* is the same for both A and B. However, there may be *better* adversaries of the forward reachability graph $\mathcal{S}_R^{\mathit{Target}}$ that assign a higher probability to reaching *Target*; hence, the maximum probability of reaching a target location in $\mathcal{S}_R^{\mathit{Target}}$ is an *upper* bound on the actual probability of reaching such a location in $\mathcal{S}_R$.

9.5 Analysis of the probabilistic steam boiler

We now present an analysis of the probabilistic steam boiler described in Section 7.4 using the algorithm **FRRectangular** from Section 9.3. The verification is carried out with respect to a reachability property which specifies that a location other than Shutdown or Emergency stop is reached after a certain amount of time. For convenience, we restrict this bound to a very low value, namely 25 time units. Given the complexity of the model, and the fact that most of the analysis given below was done by hand, it was felt that this restriction was necessary to minimise errors; nevertheless, the example demonstrates the way in which traditional hybrid automaton analysis, such as forward reachability, may be combined with probabilistic model checking to compute appropriate performance results.

Also observe that the property is somewhat artificial; a state which does not correspond to either of the locations Shutdown or Emergency stop represents safe and correct behaviour, which we would expect to be exhibited throughout the system's execution, rather than over a limited time-span. More natural choices would be a *liveness* requirement stating that the system operates correctly infinitely often, or an *invariance* property stating that a Shutdown or Emergency stop property is not reached with a certain probability or greater. While the former property is out of the scope of the reachability algorithm, the latter could be represented as the negation of a reachability property.

The reachability model checking algorithm for probabilistic rectangular automata of Section 9.3 advocates performing the verification in two steps, the first primarily concerning manipulations of the state space to obtain an appropriate forward reachability graph, and the second dealing with the computation of probabilities on this graph. We therefore divide the remainder of this section into two sections, each corresponding to one of the two verification steps.

Code	Location
F_	Off
N_	On
UF_	Urgent Off
UN_	Urgent On
FF_	Off/failure
NF_	On/failure

Table 9.1: Codes for the locations of R_{sb}.

9.5.1 Hybrid part: the forward reachability graph

We show how forward traversal through the state space of the probabilistic rectangular automaton R_{sb} shown in Figure 7.5 may be used to compute the states and transitions of a forward reachability graph that contains information that can be used to obtain a bound on the probability of the system satisfying a reachability or invariance property.

Recall that the reachability property that we wish to verify is that the system reaches a location other than Shutdown or Emergency stop after 25 time units have elapsed. This property can be naturally specified in terms of a reachability property referring to locations *only*, using the methodology of the reachability of symbolic states as given in Section 6.2.2. That is, in order to characterise the global time of system execution, we introduce a new clock denoted by c. The clock c is initially set to 0, and is not reset by any of the discrete transitions of the model. We also extend R_{sb} with a new "meta"-location Target to represent the target location. This location can be reached from any location apart from Shutdown and Emergency Stop. We associate an extra probability distribution with each such location, the support of which comprises a single edge leading to the location Target, and the enabling condition of which is $c \geq 25$. Then, the probabilistic reachability property which requires that the system is in a location other than Shutdown or Emergency stop after a period of 25 time units has elapsed since the commencement of its operation, with probability 0.98 or greater, can be expressed as the tuple $(\{\mathsf{Target}\}, \geq, 0.98)$.

For convenience, we assume that the variable representing the lower bound on the guess of the water level, denoted by g_l, is set to a constant integer value on initialisation of the model, and at each time that either of the locations Off or On are entered. Similarly, we assume that g_u is also set to some constant integer in these circumstances. This enables us to disregard the values of these variables whenever control resides in Off or On, and also in Urgent off and Urgent on. Therefore, the values of g_l and g_u are the same for all polyhedra for these locations.

The symbolic states generated by the algorithm **FRRectangular** are given in Tables B.1, B.2, B.3, B.4, B.5 and B.6 of Section B.1, and are assembled into the concurrent probabilistic system given in Figure 9.1. Each state of the system has a

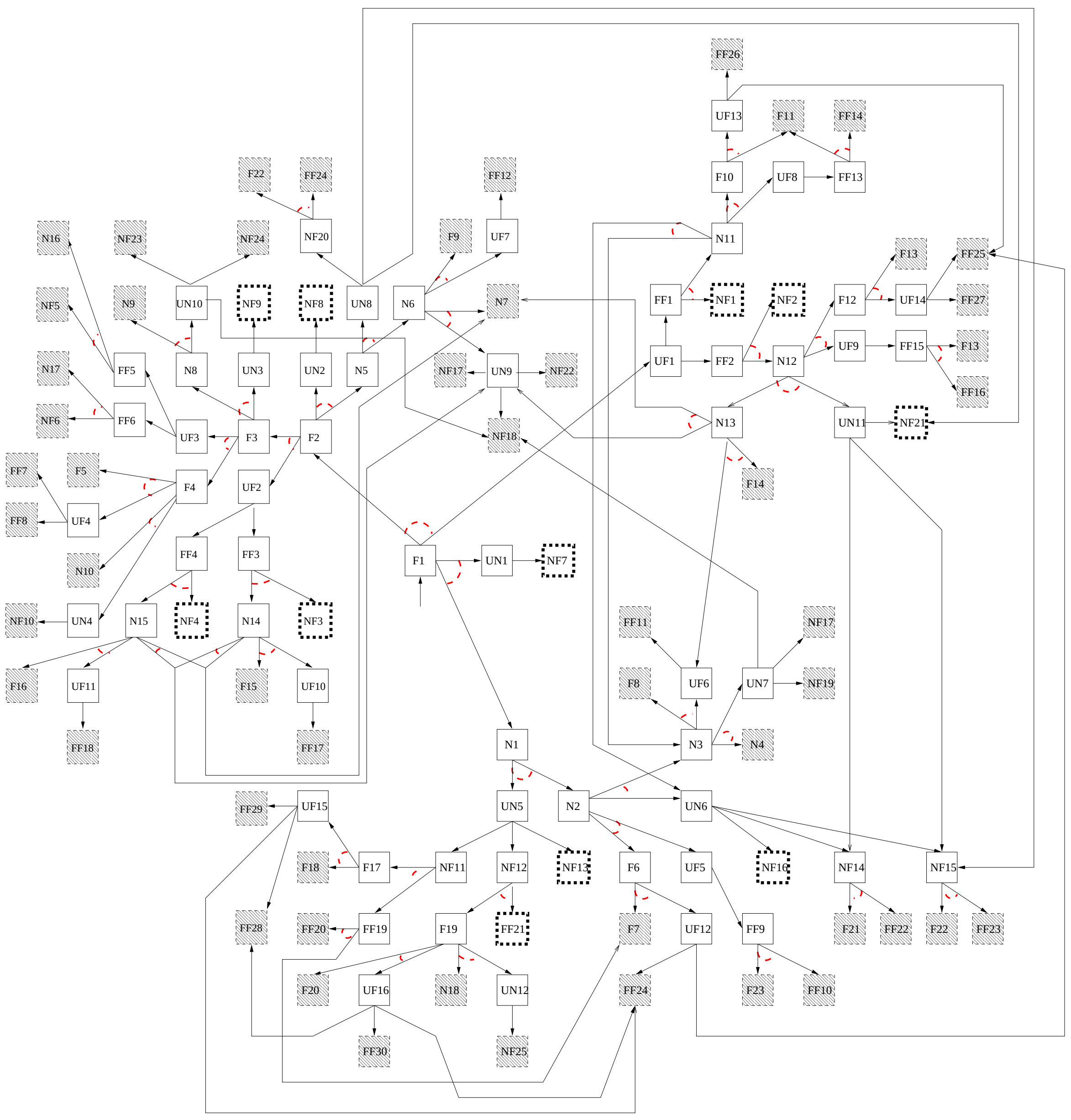


Figure 9.1: A concurrent probabilistic system for the forward reachability graph $\mathcal{S}^{Target}_{R_{sb}}$.

code, where the letters refer to the location of the probabilistic rectangular automaton according to the rules in Table 9.1, and the number uniquely identifies the polyhedron associated with that state. Note that the shaded states correspond to symbolic states for which it is possible to make a single transition to reach a target state; we omit the actual target states for simplicity. Similarly, the bold dotted states denote the symbolic states for which is possible to make a single transition to a Shutdown state.

Discrete transition	Probability
Off → Off	0.9
Off → Urgent off	0.1
Off → On	0.9
Off → Urgent on	0.1
On → On	0.9
On → Urgent on	0.1
On → Off	0.9
On → Urgent off	0.1
Off/failure → Off	0.9
Off/failure → Off/failure	0.1
Off/failure → On	0.9
Off/failure → On/failure	0.1
On/failure → On	0.9
On/failure → On/failure	0.1
On/failure → Off	0.9
On/failure → Off/failure	0.1

Table 9.2: A set of probabilities for R_{sb}.

9.5.2 Probabilistic part: computing the reachability probabilities

We now proceed to analyse the probabilistic aspects of the steam boiler model. Naturally, our first task is to specify the probabilities of the discrete transitions of the probabilistic rectangular automaton R_{sb}, which we have avoided doing so far for simplicity. For convenience, we select a very simple series of probabilities, which are shown in Table 9.2 along with the transitions to which they correspond (for example, the code Off → Urgent off denotes the discrete transition from the location Off to Urgent off).

Observe that we have omitted the probabilities for the discrete transitions from the locations Urgent off and Urgent on to Off/failure and On/failure respectively, as they are always taken with probability 1.

Next we proceed to the analysis of the concurrent probabilistic system shown in Figure 9.1. We perform a manual analysis by inspection of the diagrammatic representation of the system, the result of which is then validated through automatic verification using the tool Prism [dAKN+00].

Manual analysis

Recall that we are interested in computing the maximum probability of reaching a shaded state, which corresponds to successful operation of the system after 25 time units. From the simple fact that the number of transitions leading to such states is high, we can easily perform a manual model checking analysis of the system. That

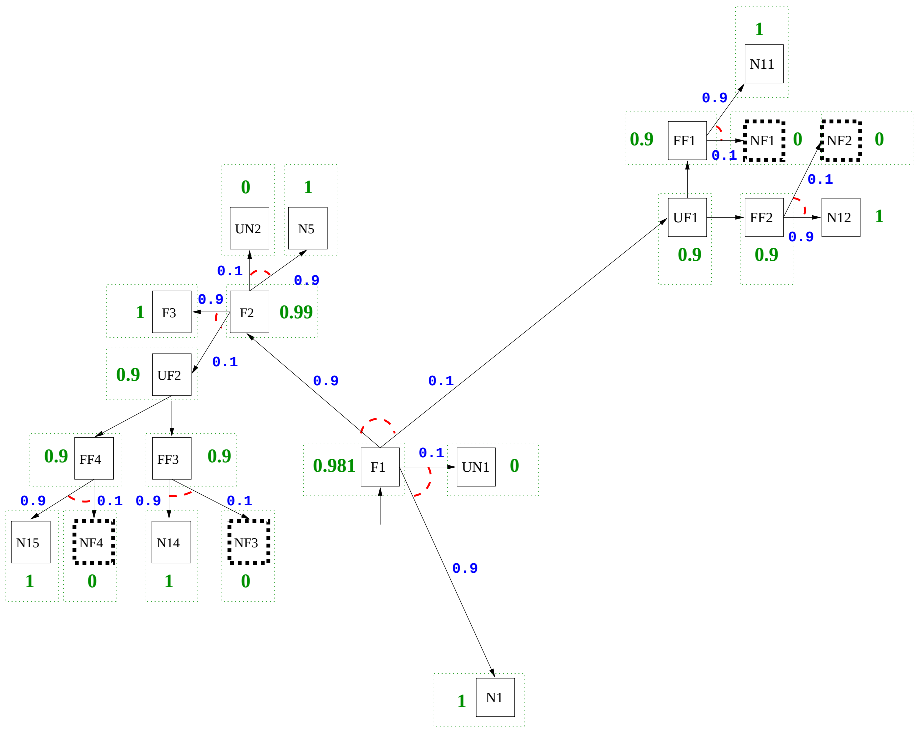


Figure 9.2: A pruned concurrent probabilistic system for the forward reachability graph $\mathcal{S}^{Target}_{R_{sb}}$.

is, in the style of the extension of the labelling algorithm of [CES86] for probabilistic-nondeterministic systems given in [BdA95], we start from the shaded states and traverse the transition relation of the system backwards. However, unlike [BdA95], because there are no loops in the system (the progress of the global clock c ensures this), we can calculate the maximal probability of reaching a shaded state "on-the-fly" as we progress from state to state. For example, consider the state F19 at the bottom-left of Figure 9.1; for both of the distributions enabled in this state, the probability of reaching a shaded state is 1, and therefore F19 can be labelled with probability 1. For its direct predecessor, the state NF12, the probability of reaching a shaded state is $(0.9 \times 1) + (0.1 \times 0) = 0.9$ (given the probabilities in Table 9.2).

Furthermore, this process allows us to *prune* the diagrammatic representation of the concurrent probabilistic system to remove extraneous states for which the maximum probability of reaching a shaded state is either 0 or 1. We perform this transformation on all of the states of the concurrent probabilistic system apart from those whose direct predecessors reach the shaded states with probability strictly between 0 or 1. The resulting concurrent probabilistic system is shown in Figure 9.2. Note that the probabilities of distributions label the appropriate edges of the graph, and the maximal probabilities of reaching a shaded state labels each state.

As the initial state F1 is labelled with the probability 0.981, we conclude that the maximal probability of reaching a shaded state is strictly greater than 0.98. Therefore, the answer to the reachability problem "with probability 0.98 or greater, the system is not in a Shutdown or Emergency stop location after 25 time units" is "MAYBE". We note that if the probability bound was greater than 0.981, then the answer would be "NO".

Automatic analysis

There are two ways in which the results of the above verification may be supported by the use of model checking tools. Firstly, the scripting mechanism of HYTECH [HHW97] may be used to output the reachable symbolic states of the probabilistic rectangular automaton in question. Secondly, the experimental model checking tool PRISM [dAKN$^+$00] may be used to verify probabilistic-nondeterministic systems against probabilistic reachability formulae, and can therefore be used to obtain the result of the probabilistic analysis. We opt to perform the latter analysis only, using as our system model the entire (not pruned) concurrent probabilistic system of the forward reachability graph $\mathcal{S}_{R_{sb}}^{Target}$.

The system description language of the tool PRISM is that of *probabilistic modules*, which is a probabilistic-nondeterministic variant of the reactive modules of Alur and Henzinger [AH99]. Concurrent probabilistic systems can be coded naturally as probabilistic modules; the way in which the forward reachability graph $\mathcal{S}_{R_{sb}}^{Target}$ is expressed in this language is presented in Section B.2.2. The language for the specification of requirements is that of PCTL [HJ94, BdA95]. Recall from Section 3.3 that this logic is equivalent to PBTL, and can therefore be used to specify reachability properties such as the one of interest in this section. As all shaded symbolic states correspond to reaching the target with probability 1, our property can be reduced to the reachability of such states. This requirement can be encoded as the PBTL property $[\mathtt{true} \exists \mathcal{U} \mathit{shaded}]_{\geq 0.98}$, where, clearly, *shaded* refers to the shaded symbolic states. Now, using the equivalences between PCTL and PBTL formulae [BK98], we can observe that the PBTL formula above can be represented alternatively as the PCTL formula $\neg \mathbb{P}_{<0.98}(\mathtt{true}\ \mathcal{U}\ \mathit{shaded})$. This property can then be coded in the requirement property syntax of PRISM (see Section B.2).

The tool PRISM can then be used to verify the forward reachability graph $\mathcal{S}_{R_{sb}}^{Target}$ against the property. An abbreviated version of the resulting output is given in Section B.2.3. As PRISM outputs a list of states along with the probabilities with which they satisfy the formula, we can verify the accuracy of the probability computed manually in the previous section. Hence, we conclude that the answer to the probabilistic reachability problem is indeed "MAYBE".

Chapter 10

Conclusions

This thesis has presented a number of formalisms and analysis techniques for probabilistic hybrid systems. The main contributions of the thesis are now summarised.

- We introduced the model of *probabilistic timed automata* in Chapter 4, which features both nondeterministic and probabilistic choice of transitions. More precisely, the discrete transitions of the model are given by discrete probability distributions, which are enabled or disabled according to the values of the model's clock variables. Issues of liveness in probabilistic timed automata were explored.

- A model checking method for verifying probabilistic timed automata against properties of a timed, probabilistic temporal logic was presented in Chapter 5. This logic can express properties such as *soft deadlines*, an example of which is the property, "given a request, a response is given within 5 seconds with probability 0.999 or greater". The verification method used the region equivalence of Alur and Dill [AD94] to obtain a finite state concurrent probabilistic system representation of the probabilistic timed automaton, which can be model-checked using established, untimed, probabilistic verification techniques.

- Chapter 6 presented a second model checking method for probabilistic timed automata. This algorithm can be used for the verification of probabilistic, timed reachability properties, such as, "with probability 0.98 or greater, the system will reach a task_completed state". The verification method was based on forward exploration through the state space of a probabilistic timed automaton from its initial state.

- The formalism of probabilistic timed automata was extended to that of *probabilistic rectangular automata* in Chapter 7. This model allows more expressive continuous behaviour to be specified than in probabilistic timed automata. An example of a probabilistic rectangular automaton to model a faulty steam boiler was given.

- Chapter 8 introduced a number of translation methods for subclasses of probabilistic rectangular automata, which allow such models to be reduced to probabilistic timed automata.

- Finally, the forward reachability algorithm of Chapter 6 was extended to the full class of probabilistic rectangular automata in Chapter 9. An example of the way in which this method may be used to verify probabilistic reachability properties of the steam boiler model in Chapter 7 was also given.

We now summarise a number of significant lessons learned in the course of the research described in this thesis. First, consider the fact that model checking methods for (non-probabilistic) timed and hybrid systems are invariably based on a reduction of a model's state set to a finitary set. For example, model checking results in the non-probabilistic context are based on the fact that (a subset of) the state set of a timed automaton is represented as a finite set of locations paired with zones, and (a subset of) the state set of a linear hybrid automaton is represented as a finite set of locations paired with polyhedra (of course, state exploration procedures for linear hybrid automata may not terminate, in which case model checking cannot return an answer). Therefore, reasoning about infinite-state models is reduced to reasoning about finite representations. We follow this paradigm in the thesis, and always perform probability computations on some finite-state representation of the probabilistic timed or rectangular automaton in question. Such finite-state representations are either obtained via a partition of the state space (as with the region graph approach of Chapter 5), or via a collection of possibly overlapping state sets obtained by iterating successor relations (the forward reachability based methods of Chapter 6 and Chapter 9). Observe the following fact: for each state in a given region of a probabilistic timed automaton, the set of PTCTL formulae satisfied in the states is the same. This result is analogous to that of [ACD93] in the non-probabilistic context, which concludes that region equivalent states satisfy the same TCTL formulae. In contrast, there may be two states encoded within a given symbolic state obtained by the forward reachability algorithms of Chapter 6 and Chapter 9 that have different maximum probabilities of reaching the target set; this leads to the fact that only an *upper bound* on the *actual* maximum probability of reaching the target set is computed by our approaches. However, in the non-probabilistic context, a target symbolic state is reached in the forward reachability graph if and only if a target state is reached in the timed automaton. Intuitively, the move from the region graph approach of Chapter 5 to the forward reachability approach in Chapter 6 and Chapter 9 has had an adverse effect on the ability to compute the required probability, as forward reachability does not preserve the branching structure necessary to compute the exact maximum reachability probability. That is, forward reachability induces an abstraction of the symbolic state space which does not contain enough information to enable the computation of the actual reachability probability (informally speaking, the abstraction that we have obtained is "too abstract" for probabilistic reachability properties). In conclusion, the state sets that are generated by a technique to analyse probabilistic timed or rectangular automata should be managed with sensitivity. The results of this thesis suggest that more computation to manipulate and store state sets is necessary in the probabilistic case than in the non-probabilistic case. For example, the forward reachability algorithms for non-probabilistic timed automata presented in [LPY97a, DT98] check whether a newly generated state set is *included within* a previous state set (or contained within a union of previously generated state sets), rather than whether it is *equal to* such a previously generated set. This type of inclusion checking is performed to reduce memory usage, and has no adverse effect on the verification of reachability properties of timed automata. However, if applied literally in the probabilistic context, such an inclusion check may have the effect of *increasing* the computed maximum probability on the forward reachability graph; therefore, the upper bound on the probability that is computed by the method of Chapter 6 becomes an even more approximate bound.

Although the results of Chapter 6 concentrated on forwards reachability through the state space of a probabilistic timed automaton, it is hoped that *backwards reachability* could be used to obtain more precise, or *exact* probabilities; an algorithm for computing the exact probability of a reachability property using only such a backwards traversal within the state space of a probabilistic timed automaton has been developed [KNS00]. This method augments the algorithm of [HNSY94] by *intersection* operations on certain state sets which are necessary for the preservation of the reachability probability. Furthermore, *selective* backwards search is advocated in [BTY97, NTY00] in addition to forwards reachability, so that information relevant to the problem in question (the satisfaction of branching-time formulae in [BTY97], and that concerning the minimal reachability time in [NTY00]) is obtained. It is hoped that such selective methods may be applied successfully to probabilistic timed automata; such an approach would then enjoy the benefits of forward reachability, in that only reachable portions of the state space are explored, and those of the backwards reachability algorithm of [KNS00], which calculates the exact probability. Similarly, an algorithm to compute the *probabilistic time-abstract bisimulation* quotient, in the style of a mix of the probabilistic bisimulation algorithm of [Bai96, BEM00], and the time-abstract bisimulation algorithm of [TY00] could prove fruitful. This is particularly important in the case of properties of the form $\forall$-until, as it is not yet clear how such properties could be verified using backwards reachability through the state space of a probabilistic timed automaton. In the non-probabilistic context, backward reachability can be used to verify a property of this form by using a reduction to a time-bounded $\forall$-until formula, which can in turn be reduced to an $\exists$-until formula. Future work could concern extending these results to the probabilistic context.

Recent work by the author of this thesis, in collaboration with Marta Kwiatkowska and Gethin Norman, has concerned the application of the forward reachability technique to the IEEE1394 (FireWire) root contention protocol. Using as a basis the work of [SV99, D'A99a, SS00], which concerned the analysis of this protocol, the tool HyTech is used to implement the forward reachability algorithm of Chapter 6, and to assemble the forward reachability graph, which is then subject to a maximum reachability computation using the tool Prism. The property to be verified is a time-bounded, "deadline" property, which expresses the requirement that contention is resolved before a certain amount of time has elapsed, with a certain probability or greater. The significant lesson learned from this case study concerned the fact that the bottleneck of the analysis lies in the (non-probabilistic) analysis of the state space of the model, as described in the forward reachability algorithm (broadly speaking, constructing a forward reachability graph would take hours invariably, followed by a reachability computation in Prism lasting seconds). These conclusions suggest that, for certain systems and properties, the option to *interleave* the forward reachability algorithm with a preliminary probability computation may be advantageous, as opposed to the current approach which considers probabilities only after the forward reachability process has terminated. For instance, a user of a (as yet hypothetical) model checking tool for probabilistic timed automata could have the option to specify at which point in the forward reachability computation (in terms of the number of generated symbolic states) a preliminary probabilistic computation could take place, and the frequency of subsequent probabilistic computations. Then, if the probability bound expressed in the reachability property in question is high, then the user may require that the first probability computation on the generated state space should only be performed when a large number of symbolic states have been generated; conversely, if the property's probability bound is low, then the user may aim for early termination of the model checking process by requiring that probability computations are performed after a relatively low number of symbolic states have been generated, and should be carried out with a high frequency.

An important characteristic of the models presented in this thesis is that their continuous components are essentially *orthogonal* to the probabilistic aspects. More precisely, although there exists some interaction between the behaviour of the continuous variables and the probabilistic transitions, as given by the enabling conditions of distributions, this is indirect. An alternative approach, taken in [HLS00], is to associate a *stochastic differential equation* with each location, which results in the continuous behaviour of the model being governed by a continuous probability distribution. Furthermore, in the context of real-time systems, the model of continuous-time Markov chains permits the modelling of delays by exponential distributions. These models do not contain nondeterministic choice, and execute transitions on the basis of probabilities alone. Therefore, their disadvantage is that they do not allow for "under-specification"; that is, in models such as probabilistic timed or rectangular automata, nondeterminism could be used to model aspects of the design for which only partial information is available. This may be particularly appropriate in the case of such complex probabilistic hybrid systems that we intend in the future to model (such as the hybrid systems mentioned in Chapter 1), and was one of the motivations of de Alfaro when introducing stochastic transition systems [dA97a, dA98b], for which it is possible to leave the rates of any number of exponential distributions of the model unspecified.

Observe that [HLS00] presents a reachability method for their stochastic hybrid automaton model, and model checking methods for stochastic transition systems and continuous-time Markov chains are given in [dA97a, dA98b] and [BKH99, BHHK00] respectively. Similarly, Kwiatkowska et al. [KNSS00b] present a model checking method for probabilistic timed automata for which clocks may be reset according to continuous probability distributions (these distributions must have bounded support with integer endpoints). However, the work of de Alfaro does not consider *deadline properties*; the model checking method for deadline properties of [BKH99, BHHK00] is approximate; and the model checking approach of [KNSS00b] is approximate for all properties, and works by *discretisation* of continuous probability distributions during construction of a region graph of the system. The observations suggest that we could approximate continuous probability distributions using probabilistic timed automata; that is, we could represent such a distribution with a discrete distribution over a number of edges, each of which results in a clock being set to some value, or within some bounds (where the exact value within those bounds could be chosen nondeterministically). Some of these thoughts are inspired by a personal communication from Pedro D'Argenio [D'A00b].

Another approach, which addresses a number of fundamental issues concerning continuous-state probabilistic systems, is that of Desharnais, Gupta, Jagadeesan and Panangaden [DGJP00]. Their aim is to construct *approximative* finite-state Markov chains which simulate the original system (their definition of simulation differs from that used in this thesis). Naturally, these results are of great interest and relevance in the context of work on probabilistic hybrid systems, and should be investigated further.

The requirement of *initialisation* on probabilistic stopwatch, multisingular and rectangular automata, which is needed to allow a translation to probabilistic timed automata in Chapter 8, is a problematic restriction. The steam boiler of Chapter 7 cannot be modelled by an initialised model; for example, we must know the exact height of the water level to know how much time it may take before reaching a dangerous lower level if we switch the pump off. Furthermore, the fact that rectangular flow conditions and resets are used weakens the correspondence between probabilistic rectangular automata and probabilistic timed automata, and we only have the result that the latter simulates the former. We note that, in the non-probabilistic context, there exists another relation in the opposite direction: the timed automaton constructed from a rectangular automaton *backward simulates* the latter model [HKPV98, LV96]. Then we have the result that the two models accept the same finite languages, which [HKPV98] extends to infinite languages. Although it means that LTL is preserved by this translation [HM00b, AHLP00], the result of this translation is the loss of the branching structure of the paths. Maintaining this branching structure is a key attribute when attempting to transfer results from the non-probabilistic to probabilistic context, as our atomic element that we use for analysis is an adversary, which, of course, has a probabilistically branching structure. Furthermore, it is not clear how a concept such as backwards simulation applies to probabilistic systems.

The forward reachability algorithm (for both probabilistic timed and rectangular automata) is inefficient, as it disregards information regarding probability that could be maintained during the forward traversal; in its current form, the actual probabilities are not considered until after the forward search has terminated. We note that [BFH$^+$01] advocates *branch and bound* methods for the verification of priced timed automata (see Section 2.3.4); that is, the forward reachability search is guided by the current computed prices associated with its paths (of regions). For example, if the price of one path rises above that of another, then we explore the latter, until that exceeds the cost of another path, which we then explore, and so on. This method could be adapted to the probabilistic case, although it is likely to be more involved. Nevertheless, this could provide an interesting avenue of research if the overhead from the probability computation is not excessive.

Finally, we note that the notions of progress presented in Section 4.4 could be treated in an algorithmic manner. For the method of quantification over the subset of progressive adversaries to be practical, we either require a technique to identify automatically the appropriate subset, or to have a model checking technique which considers only adversaries in this subset. Although the latter strategy is adopted in the context of probabilistic-nondeterministic transition systems with a notion of *expected time* in [dA97a], a different approach is traditionally taken for timed automata [HNSY94, Tri99b]. This involves the characterisation of deadlocks and timelocks as reachability or logical properties, which can then be verified. If the system fails the property, then the system designer can take appropriate action to remove such modelling errors. In particular, the backwards reachability model checking algorithm of [HNSY94] expresses time progress as a timed μ-calculus formula ϕ_{nz} (where nz stands for "non-zeno"). Because the algorithm computes the "characteristic set" of states that satisfy a timed μ-calculus formula, the invariant conditions of the timed automaton in question can be strengthened to make all states not satisfying ϕ_{nz}, and therefore not allowing time progress, be inadmissible. Similarly, Tripakis presents forward reachability algorithms for deadlock and timelock detection [Tri98, Tri99b]. The extension of such techniques to probabilistic timed automata is of great interest, although it is not clear whether the probabilistic nature of progress in this context will add significant complications to this work.

Appendix A

Proofs

A.1 Introduction

We recall the theorems from Chapter 3 which we wish to prove.

Theorem 3.3.5 *Let $\mathcal{S}$ be a concurrent probabilistic system, let $\mathcal{A} \subseteq Adv$ be a set of adversaries of $\mathcal{S}$, and let Φ be a $\forall$PBTL formula. If $s_1 \preceq s_2$, for $s_1, s_2 \in S$, then $s_2 \models_{\mathcal{A}} \Phi$ implies $s_1 \models_{\mathcal{A}} \Phi$.*

Theorem 3.3.6 *Let $\mathcal{S}$ be a concurrent probabilistic system, let $\mathcal{A} \subseteq Adv$ be a set of adversaries of $\mathcal{S}$, and let Φ be a PBTL formula. If $s_1 \simeq s_2$, for $s_1, s_2 \in S$, then $s_1 \models_{\mathcal{A}} \Phi$ if and only if $s_2 \models_{\mathcal{A}} \Phi$.*

The proofs of both Theorem 3.3.5 and Theorem 3.3.6 proceed by induction on the subformulae of Φ, with the non-trivial case concerning the until formulae $[\Phi_1 \forall\mathcal{U} \Phi_2]_{\sqsupseteq\lambda}$, $[\Phi_1 \exists\mathcal{U} \Phi_2]_{\sqsupseteq\lambda}$. The case for this type of formula is addressed in the following manner. For any adversary A of $\mathcal{S}$, we consider the set of paths of A that start from the state s_1. Then we construct an adversary B of $\mathcal{S}$ by letting B "copy" the behaviour of A featured in the aforementioned set of paths in much the same manner as in the adversary construction process of Proposition 9.4.4. We restrict our attention to the paths of B that commence in the state s_2; the adversary B can be arbitrary elsewhere. It then transpires that the pointed Markov chains $\mathrm{PMC}^A = (\mathrm{MC}^A, s_1)$ and $\mathrm{PMC}^B = (\mathrm{MC}^B, s_2)$ induced by the Markov chains of A and B, with the start states of s_1 and s_2 respectively, are such that the probability of satisfying the until path formula $\Phi_1 \mathcal{U} \Phi_2$ in PMC^A is related to the corresponding probability in PMC^B. More precisely, if the relation between s_1 and s_2 is a simulation, written $s_1 \preceq s_2$, then the probability of satisfying $\Phi_1 \mathcal{U} \Phi_2$ is at least as great in PMC^A as in PMC^B. However, if the relation between s_1 and s_2 is a bisimulation, written $s_1 \simeq s_2$, then the probability of satisfying $\Phi_1 \mathcal{U} \Phi_2$ is equal in PMC^A and PMC^B.

Most of the technical machinery required for these two proofs is common to both. We show how to construct inductively the *randomised* adversary B from any deterministic adversary A of $\mathcal{S}$. This construction process proceeds inductively along the length of paths from the states s_1 and s_2; for each $i \in \mathbb{N}$, we take the choices of event-distribution pairs made by A after paths starting in s_1 of length i, and use these to define the choices of B after paths starting in s_2 also of length i. This section is essentially a presentation of the "execution correspondence lemma" of [Seg95] in a less general context.

A.2 Adversary construction process

A.2.1 Preliminaries

We now turn our attention to the method by which, for any deterministic adversary A, a randomised adversary B can be constructed. As stated in Section A.1, we proceed on the length of paths from s_1 and s_2, defining the choices of B with the following dual criteria:

Criterion 1 the choices of B after paths from s_2 of a certain length emulate the choices of A along paths from s_1 of the same length, and

Criterion 2 the functions $Prob^A_{fin}$ and $Prob^B_{fin}$ induce distributions over paths from s_1 and s_2 respectively which are related via a weight function with respect to a special relation on paths.

Both of these points require clarification. With regard to Criterion 1, recall that the choice of the deterministic adversary A after a path will comprise an event-distribution pair; then, for the choice of B to emulate this choice, we require that, for all event-distribution pairs to which it assigns positive probability, both (a) the event component is equal to the event component of the choice of A, and (b) the distribution component is related to the distribution component of the choice of A via a simulation relation $\preceq$ or the bisimulation relation $\simeq$. As in the adversary construction process of Proposition 9.4.4, we proceed by defining a countably infinite family of randomised adversaries $\langle B_i \rangle_{i\in\mathbb{N}}$, where the randomised adversary B is obtained as i approaches the limit. Each B_i defines the choices of B after paths of length i, acts like B_{i-1} for all paths of length strictly less than i, and can be arbitrary for all other paths.

With regard to Criterion 2 on the choices of B, the relation on paths is such that two paths are related by the new relation if and only if (a) they are of the same length, and (b) each state along the length of one path is related via a simulation relation $\preceq$ or the bisimulation relation $\simeq$ to the state at the corresponding point along the length of the other path. In a similar manner to Definition 9.4.2, we call this relation a *path relation*.

Definition A.2.1 (Path relation) *For a given concurrent probabilistic system $\mathcal{S}$, and a relation on states $\mathcal{R} \in S \times S$, the* path relation $\Re \subseteq \mathit{Path}_{\mathit{fin}} \times \mathit{Path}_{\mathit{fin}}$ *is such that, for every finite path $\omega, \pi \in \mathit{Path}_{\mathit{fin}}$, we have $\omega \Re \pi$ if and only if:*

1. $|\omega| = |\pi|$,
2. $\mathit{last}(\omega)\mathcal{R}\mathit{last}(\pi)$, *and*
3. *if* $|\omega| = |\pi| \geq 1$, *then* $\omega^{(i-1)} \Re \pi^{(i-1)}$ *for* $i = |\omega| = |\pi|$.

Observe that clause (3) in Definition A.2.1 requires that the immediate prefixes of ω and π (that is, the prefixes obtained by simply removing the final transition of ω and π) are also related via $\Re$. Applying this intuition recursively yields the fact that, if two paths are such that $\omega \Re \pi$, then each state along the length of ω is related via $\mathcal{R}$ to the state at the corresponding point along the length of π.

In order to reason about the probability of the paths of both A and B, we use two countably infinite families of path distributions $\langle \mathbf{p}_i^A \rangle_{i \geq 1}$, $\langle \mathbf{p}_i^B \rangle_{i \geq 1}$. For each $i \geq 1$, the distribution $\mathbf{p}_i^A$ is derived from the probabilities assigned by $\mathit{Prob}_{\mathit{fin}}^A$ to the finite paths of A of length i such that their first state is s_1. Similarly, for each $i \geq 1$, the distribution $\mathbf{p}_i^B$ is derived from the probabilities assigned by $\mathit{Prob}_{\mathit{fin}}^{B_{i-1}}$ to the finite paths of B_{i-1} of length i such that their first state is s_2 (again, it is useful to recall that the adversary B_{i-1} defines paths of length i). The formal definition of path distributions is given in Definition 9.4.3.

Our aim is then to show that, for each $i \geq 1$, there exists a weight function $\mathbf{w}_i$ for $(\mathbf{p}_i^A, \mathbf{p}_i^B)$ with respect to $\Re$. The existence of $\mathbf{w}_i$ then permits us to define the next choices of B (that is, the choices of the adversary B_i) *and* provides us with the appropriate mechanism to reason about the probability of the satisfaction of a until path formula for the paths of A and B from s_1 and s_2 respectively, as explained in Section A.3.

A.2.2 Construction

We commence by stating formally our aims in the construction process.

Proposition A.2.2 (Adversary construction) *Let $\mathcal{S}$ be a concurrent probabilistic system, let A be a deterministic adversary of $\mathcal{S}$, and let $s_1, s_2 \in S$ be states such that $s_1 \mathcal{R} s_2$, where $\mathcal{R} \in \{\preceq, \simeq\}$. Then there exists a countably infinite family of randomised adversaries $\langle B_i \rangle_{i \in \mathbb{N}}$ such that $\mathbf{p}_i^A \Re \mathbf{p}_i^B$ for each $i \geq 1$.*

Proof. We proceed inductively on the length of paths in $\mathit{Path}_{\mathit{fin}}^A(s_1)$, which also coincides with induction on the index of the adversaries in the family $\langle B_i \rangle_{i \in \mathbb{N}}$.

Base. Our aim is to derive the randomised adversary B_0 from the paths of length 1 of A. Let $(\sigma, p) \in \mathit{Steps}(s_1)$ be the event-distribution pair such that $A(s_1)(\sigma, p) = 1$. Then, from $s_1 \mathcal{R} s_2$, and from either the definition of $\preceq$ (Definition 3.2.15) or the definition of $\simeq$ (Definition 3.2.16), depending on $\mathcal{R}$, there exists an event-distribution pair $(\sigma, p') \in \mathit{Steps}(s_2)$ such that $p \mathcal{R} p'$. We simply let $B_0(s_2)(\sigma, p') = 1$; the adversary B_0 can be arbitrary for all other paths.

We now show that this choice of B_0 results in the preservation of the induction hypothesis; that is, $\mathbf{p}_1^A \Re \mathbf{p}_1^B$. Recall that $\mathbf{p}_1^A \Re \mathbf{p}_1^B$ if and only if there exists a weight function $\mathbf{w}_1$ for $(\mathbf{p}_1^A, \mathbf{p}_1^B)$ with respect to $\Re$. We show that $\mathbf{w}_1$ can be obtained from a weight function for (p, p') with respect to $\mathcal{R}$, which exists by $p \mathcal{R} p'$. First, we must identify the paths in $\mathit{Path}_1^A(s_1), \mathit{Path}_1^{B_0}(s_2)$ which are related by $\Re$. Recall the definition of $\Re$ (Definition A.2.1). Condition (1) is trivially satisfied by all paths in $\mathit{Path}_1^A(s_1), \mathit{Path}_1^{B_0}(s_2)$, as all such paths are of length 1. Observe that, when viewing s_1, s_2 as paths of length 0, we have that $s_1 \mathcal{R} s_2$ implies $s_1 \Re s_2$. Hence, condition (3) is also satisfied by all paths $\omega \in \mathit{Path}_1^A(s_1), \pi \in \mathit{Path}_1^{B_0}(s_2)$, because their immediate prefixes $\omega^{(0)} = s_1$ and $\pi^{(0)} = s_2$ are such that $\omega^{(0)} \Re \pi^{(0)}$. Therefore, as only condition (2) of Definition A.2.1 remains, for any paths $\omega \in \mathit{Path}_1^A(s_1), \pi \in \mathit{Path}_1^{B_0}(s_2)$, we have $\omega \Re \pi$ if and only if $\mathit{last}(\omega) \mathcal{R} \mathit{last}(\pi)$. Furthermore, if w is a weight function for (p, p') with respect to $\mathcal{R}$, then for $\omega \in \mathit{Path}_1^A(s_1), \pi \in \mathit{Path}_1^{B_0}(s_2)$ where $\omega = s_1 \xrightarrow{\sigma,p} s$ and $\pi = s_2 \xrightarrow{\sigma,p'} s'$, let $\mathbf{w}_1(\omega, \pi) = w(s, s')$.

We verify that $\mathbf{w}_1$ is indeed a weight function for $(\mathbf{p}_1^A, \mathbf{p}_1^B)$ with respect to $\Re$. Recall the definition of weight functions (Definition 3.2.12). Observe that, for $\omega \in \mathit{Path}_1^A(s_1), \pi \in \mathit{Path}_1^{B_0}(s_2)$ such that $\omega = s_1 \xrightarrow{\sigma,p} s$ and $\pi = s_2 \xrightarrow{\sigma,p'} s'$, we have $\mathbf{w}_1(\omega, \pi) > 0$ if and only if $w(s, s') > 0$. Also recall that $w(s, s') > 0$ implies $s \mathcal{R} s'$. As we established in the previous paragraph, the statements $\omega \Re \pi$ and $s \mathcal{R} s'$ are equivalent for paths ω, π of length 1, and therefore $\mathbf{w}_1(\omega, \pi) > 0$ implies $\omega \Re \pi$. This satisfies the first condition in Definition 3.2.12. For the second condition, consider any path $\omega \in \mathit{Path}_1^A(s_1)$ and the following sum:

$$\begin{aligned}
& \sum_{\pi \in \mathit{Path}_1^{B_0}(s_2)} \mathbf{w}_1(\omega, \pi) \\
= & \sum_{s' \in \mathsf{support}(p')} w(\mathit{last}(\omega), s') && \text{(by definition of } \mathbf{w}_1 \text{ and } \mathit{Path}_1^{B_0}(s_2)) \\
= & \; p(\mathit{last}(\omega)) && \text{(by Definition 3.2.12)} \\
= & \; \mathbf{p}_1^A(\omega) && \text{(by Definition 3.2.2, Definition 9.4.3).}
\end{aligned}$$

Similarly (using the same manipulations and justifications), for any $\pi \in \mathit{Path}_1^{B_0}(s_2)$, we have:

$$\sum_{\omega \in \mathit{Path}_1^A(s_1)} \mathbf{w}_1(\omega, \pi) = \mathbf{p}_1^B(\pi) \; .$$

Therefore, $\mathbf{w}_1$ is a weight function, and hence $\mathbf{p}_1^A \Re \mathbf{p}_1^B$.

Induction. Consider any $i \geq 1$. By induction, we assume the existence of a randomised adversary B_{i-1}, which is such that there exists a weight function $\mathbf{w}_i$ for $(\mathbf{p}_i^A, \mathbf{p}_i^B)$ with respect to $\Re$. We now detail the construction of the randomised adversary B_i, and show that B_i is such that $\mathbf{p}_{i+1}^A \Re \mathbf{p}_{i+1}^B$.

Consider the paths $\omega \in Path_i^A(s_1)$ and $\pi \in Path_i^{B_{i-1}}(s_2)$, and let:

$$\mathsf{Random}_i(\omega, \pi) = \frac{\mathbf{w}_i(\omega, \pi)}{\mathbf{p}_i^B(\pi)} .$$

Now consider the single path $\pi \in Path_i^{B_{i-1}}(s_2)$; we define the randomised choice of the adversary B_i after π in the following way. Let

$$Path_{\mathbf{w}_i,\pi}^A(s_1) = \{\omega \in Path_i^A(s_1) \mid \mathbf{w}_i(\omega, \pi) > 0\}$$

be the set of paths of A of length i such that $\mathbf{w}_i$ assigns a positive weight to the each of the paths $\omega \in Path_{\mathbf{w}_i,\pi}^A(s_1)$ and π. From the fact that $\mathbf{w}_i$ is a weight function with respect to $\Re$, for every $\omega \in Path_{\mathbf{w}_i,\pi}^A(s_1)$, we have $\omega \Re \pi$; and, by the definition of $\Re$ (Definition A.2.1) we have $last(\omega) \mathcal{R} last(\pi)$. Therefore, as $\mathcal{R} \in \{\preceq, \simeq\}$, and by the definition of $\preceq$ (Definition 3.2.15) and $\simeq$ (Definition 3.2.16), if $A(\omega)(\sigma, p) = 1$ for the event-distribution pair $(\sigma, p) \in Steps(last(\omega))$, then there exists at least one event-distribution pair $(\sigma, p') \in Steps(last(\pi))$ such that $p \mathcal{R} p'$. We select arbitrarily one such pair, and denote it by $\mathsf{Choice}_i(\omega, \pi)$. Note that, for distinct paths $\omega, \omega' \in Path_{\mathbf{w}_i,\pi}^A(s_1)$, we may have $\mathsf{Choice}_i(\omega, \pi) = \mathsf{Choice}_i(\omega', \pi)$. For each $(\sigma, p) \in Steps(last(\pi))$, let

$$Path_{\sigma,p}^A(s_1) = \{\omega \in Path_i^A(s_1) \mid \mathsf{Choice}_i(\omega, \pi) = (\sigma, p)\}$$

be the set of paths of A of length i such that the event-distribution pair $(\sigma, p) \in Steps(last(\pi))$ is used after π to copy the event-distribution of each of the paths. Then, for any $(\sigma, p) \in Steps(last(\pi))$, we construct $B_i(\pi)(\sigma, p)$ by summing the values of $\mathsf{Random}_i(\omega, \pi)$ for paths in the set $Path_{\sigma,p}^A(s_1)$:

$$B_i(\pi)(\sigma, p) = \sum_{\omega \in Path_{\sigma,p}^A(s_1)} \mathsf{Random}_i(\omega, \pi) .$$

Note that $Path_{\sigma,p}^A(s_1) = \emptyset$ for any $(\sigma, p) \in Steps(last(\pi))$ such that there does not exist some $\omega \in Path_{\mathbf{w}_i,\pi}^A(s_1)$ such that $(\sigma, p) = \mathsf{Choice}_i(\omega, \pi)$. Therefore, $B_i(\pi)(\sigma, p) = 0$ in such a case.

We now verify that $B_i(\pi)$ is a distribution over the set $Steps(last(\pi))$. Firstly, noting that, for any $\pi \in Path_i^{B_{i-1}}(s_2)$, we have $\sum_{\omega \in Path_1^A(s_1)} \mathbf{w}_i(\omega, \pi) = \mathbf{p}_i^B(\pi)$, and for any $\omega \in Path_i^A(s_1)$, we also have $0 \leq \mathbf{w}_i(\omega, \pi) \leq \mathbf{p}_i^B(\pi) \leq 1$, then we must have $B_i(\pi)(\sigma, p) \in [0, 1]$ for any $(\sigma, p) \in Steps(last(\pi))$. Secondly, we show that:

$$\sum_{(\sigma,p) \in Steps(last(\pi))} B_i(\pi)(\sigma, p) = 1 .$$

Observe the following fact.

Fact A.2.3 *Let A, B_i be the (deterministic and randomised, respectively) adversaries, $s_1, s_2 \in S$ the states, $\pi \in Path_i^{B_{i-1}}(s_2)$ the path, and $\mathbf{w}_i$ the weight function defined above. Then the set of sets*

$$\{Path^A_{\sigma,p}(s_1) \mid (\sigma, p) \in Steps(last(\pi))\}$$

is an $\emptyset$-inclusive partition of $Path^A_{\mathbf{w}_i,\pi}(s_1)$.

We now justify Fact A.2.3. Firstly, assume by contradiction that there exists the path $\omega \in Path^A_{\sigma,p}(s_1) \cap Path^A_{\sigma',p'}(s_1)$ for distinct pairs $(\sigma, p), (\sigma', p') \in Steps(last(\pi))$. Then, by the construction of $B_i(\pi)$, we have $(\sigma, p) = \mathsf{Choice}_i(\omega, \pi)$ and $(\sigma', p') = \mathsf{Choice}_i(\omega, \pi)$: a contradiction by our definition of Choice_i. Therefore, by the construction of $B_i(\pi)$, it must be the case that the sets $Path^A_{\sigma,p}(s_1)$ for each $(\sigma, p) \in Steps(last(\pi))$ are pairwise disjoint. Secondly, by the construction of $B_i(\pi)$, for every path $\omega \in Path^A_{\mathbf{w}_i,\pi}(s_1)$, we have defined $\mathsf{Choice}_i(\omega, \pi) \in Steps(last(\pi))$. Therefore:

$$Path^A_{\mathbf{w}_i,\pi}(s_1) = \bigcup_{(\sigma,p)\in Steps(last(\pi))} Path^A_{\sigma,p}(s_1) \; ,$$

and we have shown Fact A.2.3. Finally, to show that $B_i(\pi)$ is a distribution over $Steps(last(\pi))$, we observe the following:

$$\begin{aligned}
& \sum_{(\sigma,p)\in Steps(last(\pi))} B_i(\pi)(\sigma,p) \\
= & \sum_{(\sigma,p)\in Steps(last(\pi))} \sum_{\omega\in Path^A_{\sigma,p}(s_1)} \mathsf{Random}_i(\omega,\pi) && \text{(by the definition of } B_i(\pi)\text{)} \\
= & \sum_{\omega\in Path^A_{\mathbf{w}_i,\pi}(s_1)} \mathsf{Random}_i(\omega,\pi) && \text{(by Equation 3.1 and Fact A.2.3)} \\
= & \sum_{\omega\in Path^A_{\mathbf{w}_i,\pi}(s_1)} \frac{\mathbf{w}_i(\omega,\pi)}{\mathbf{p}_i^B(\pi)} && \text{(by definition of } \mathsf{Random}_i(\omega,\pi)\text{)} \\
= & \sum_{\omega\in Path^A_i(s_1)} \frac{\mathbf{w}_i(\omega,\pi)}{\mathbf{p}_i^B(\pi)} && \text{(by definition of } Path^A_{\mathbf{w}_i,\pi}(s_1)\text{)} \\
= & \frac{1}{\mathbf{p}_i^B(\pi)} \sum_{\omega\in Path^A_i(s_1)} \mathbf{w}_i(\omega,\pi) \\
= & \frac{\mathbf{p}_i^B(\pi)}{\mathbf{p}_i^B(\pi)} && \text{(by Definition 3.2.12)} \\
= & 1 \, .
\end{aligned}$$

Hence, $B_i(\pi)$ is indeed a distribution over the set $Steps(last(\pi))$, as required.

Repeating this process for all paths $\pi \in Path_i^{B_{i-1}}(s_2)$ results in a set of distributions over event-distribution pairs, each denoted by $B_i(\pi)$. Furthermore, for paths of length less than i, we let the randomised choices of B_i equal the randomised choices of B_{i-1}

for these paths; that is, for each path $\pi \in \bigcup_{1 \leq j < i} Path_j^{B_{j-1}}(s_2)$, we let $B_i(\pi) = B_{i-1}(\pi)$. The choices of B_i for all other paths is arbitrary. We have concluded our definition of the randomised adversary B_i.

We now show that B_i is such that the induction hypothesis is preserved; that is, $\mathbf{p}_{i+1}^A \Re \mathbf{p}_{i+1}^B$. Recall that $\mathbf{p}_{i+1}^A \Re \mathbf{p}_{i+1}^B$ if and only if there exists a weight function $\mathbf{w}_{i+1}$ for $(\mathbf{p}_{i+1}^A, \mathbf{p}_{i+1}^B)$ with respect to $\Re$. Our aim is therefore to show the existence of $\mathbf{w}_{i+1}$. We introduce the following notation. To simplify matters, as in the proof of Proposition 9.4.4, we express the distributions over states chosen after paths of length i as distributions over *paths*. For the path $\omega \in Path_i^A(s_1)$, let $\mathbf{p}_\omega \in \mu(Path_{i+1}^A(s_1))$ be the distribution over paths defined in the following way. For the event-distribution pair $(\sigma, p) \in Steps(last(\omega))$ which is such that $A(\omega)(\sigma, p) = 1$, we let

$$\mathbf{p}_\omega(\omega') = \begin{cases} p(s) & \text{if } \omega' = \omega \xrightarrow{\sigma,p} s \\ 0 & \text{otherwise.} \end{cases}$$

Similarly, for the paths $\omega \in Path_i^A(s_1)$, $\pi \in Path_i^{B_{i-1}}(s_2)$ which are such that $\mathbf{w}_i(\omega, \pi) > 0$, let $\mathbf{p}_{\omega\pi} \in \mu(Path_{i+1}^{B_i}(s_2))$ be the distribution over paths defined as follows. For the event-distribution pair $(\sigma, p') \in Steps(last(\pi))$ which is such that $(\sigma, p') = \mathsf{Choice}_i(\omega, \pi)$, we let

$$\mathbf{p}_{\omega\pi}(\pi') = \begin{cases} p'(s) & \text{if } \pi' = \pi \xrightarrow{\sigma,p'} s \\ 0 & \text{otherwise.} \end{cases}$$

Note that, for distinct $\omega, \omega' \in Path_i^A(s_1)$, it may be the case that $\mathsf{Choice}_i(\omega, \pi) = \mathsf{Choice}_i(\omega', \pi)$, and therefore $\mathbf{p}_{\omega\pi} = \mathbf{p}_{\omega'\pi}$; however, this fact does not complicate the following analysis.

Recall that, by the definition of B_i, for the paths ω, π and the distributions on states p, p' defined above, we have $p\mathcal{R}p'$, which denotes the existence of a weight function w for (p, p') with respect to $\mathcal{R}$. Also recall that the paths ω, π are such that $\mathbf{w}_i(\omega, \pi) > 0$. We define the function $\mathbf{w}_{\omega\pi} : Path_{i+1}^A(s_1) \times Path_{i+1}^{B_i}(s_2) \to [0,1]$ in the following manner: for $\omega \xrightarrow{\sigma,p} s \in Path_{i+1}^A(s_1)$, $\pi \xrightarrow{\sigma,p'} s' \in Path_{i+1}^{B_i}(s_2)$, simply let $\mathbf{w}_{\omega\pi}(\omega', \pi') = w(s, s')$. We show that $\mathbf{w}_{\omega\pi}$ is a weight function for $(\mathbf{p}_\omega, \mathbf{p}_{\omega\pi})$ with respect to $\Re$.

Firstly, we require that $\mathbf{w}_{\omega\pi}(\omega', \pi') > 0$ implies that $\omega' \Re \pi'$ for $\omega' \in Path_{i+1}^A(s_1)$, $\pi' \in Path_{i+1}^{B_i}(s_2)$. Clearly, $|\omega'| = |\pi'| = i + 1$, and $\mathbf{w}_i(\omega, \pi) > 0$ implies that $\omega \Re \pi$, thus satisfying the first and third requirements in the definition of $\Re$ (Definition A.2.1). Now observe that $\mathbf{w}_{\omega\pi}(\omega', \pi') > 0$ if and only if $w(last(\omega'), last(\pi')) > 0$, and that $w(last(\omega'), last(\pi')) > 0$ implies that $last(\omega')\mathcal{R}last(\pi')$. This satisfies the second condition in the definition of $\Re$. Therefore, $\mathbf{w}_{\omega\pi}(\omega', \pi') > 0$ implies that $\omega' \Re \pi'$.

Secondly, we show that

$$\sum_{\pi' \in Path_{i+1}^{B_i}(s_2)} \mathbf{w}_{\omega\pi}(\omega', \pi') = \mathbf{p}_\omega(\omega')$$

for any $\omega' \in \mathit{Path}^A_{i+1}(s_1)$. Recall that, for $\omega \in \mathit{Path}^A_i(s_1)$, the event-distribution pair $(\sigma, p) \in \mathit{Steps}(\mathit{last}(\omega))$ such that $A(\omega)(\sigma, p) = 1$ is fixed, and for $\pi \in \mathit{Path}^{B_{i-1}}_i(s_2)$, the event-distribution pair $(\sigma, p') \in \mathit{Steps}(\mathit{last}(\pi))$ such that $(\sigma, p') = \mathsf{Choice}_i(\omega, \pi)$ is also fixed. By the definition of $\mathbf{w}_{\omega\pi}$, and as the paths ω, ω', π and the distribution p' are fixed, we can write

$$\begin{aligned}
\sum_{\pi' \in \mathit{Path}^{B_i}_{i+1}(s_2)} \mathbf{w}_{\omega\pi}(\omega', \pi') &= \sum_{(\pi \xrightarrow{\sigma,p'} s) \in \mathit{Path}^{B_i}_{i+1}(s_2)} \mathbf{w}_{\omega\pi}(\omega', (\pi \xrightarrow{\sigma,p'} s)) \\
&= \sum_{(\pi \xrightarrow{\sigma,p'} s) \in \mathit{Path}^{B_i}_{i+1}(s_2)} w(\mathit{last}(\omega'), s) \\
&= \sum_{s \in \mathsf{support}(p')} w(\mathit{last}(\omega'), s) \\
&\qquad\qquad \text{(by definition of paths)} \\
&= \sum_{s \in S} w(\mathit{last}(\omega'), s) \\
&\qquad\qquad \text{(by Definition 3.2.12)} \\
&= p(\mathit{last}(\omega')) \\
&\qquad\qquad \text{(by Definition 3.2.12)} \\
&= \mathbf{p}_\omega(\omega') \\
&\qquad\qquad \text{(by definition of } \mathbf{p}_\omega\text{).}
\end{aligned}$$

Similarly, we show that

$$\sum_{\omega' \in \mathit{Path}^A_{i+1}(s_1)} \mathbf{w}_{\omega\pi}(\omega', \pi') = \mathbf{p}_{\omega\pi}(\pi')$$

for any $\pi' \in \mathit{Path}^{B_i}_{i+1}(s_2)$. By the definition of $\mathbf{w}_{\omega\pi}$, and as the paths ω, π, π' and the distribution p are fixed, we can write

$$\begin{aligned}
\sum_{\omega' \in \mathit{Path}^A_{i+1}(s_1)} \mathbf{w}_{\omega\pi}(\omega', \pi') &= \sum_{\omega' \in \mathit{Path}^A_{i+1}(s_1)} w(\mathit{last}(\omega'), \mathit{last}(\pi')) \\
&= \sum_{(\omega \xrightarrow{\sigma,p} s) \in \mathit{Path}^A_{i+1}(s_1)} w(s, \mathit{last}(\pi')) \\
&= \sum_{s \in \mathsf{support}(p)} w(s, \mathit{last}(\pi')) \\
&\qquad\qquad \text{(by definition of paths)} \\
&= \sum_{s \in S} w(s, \mathit{last}(\pi')) \\
&\qquad\qquad \text{(by Definition 3.2.12)} \\
&= p'(\mathit{last}(\pi')) \\
&\qquad\qquad \text{(by Definition 3.2.12)} \\
&= \mathbf{p}_{\omega\pi}(\pi') \\
&\qquad\qquad \text{(by definition of } \mathbf{p}_{\omega\pi}\text{).}
\end{aligned}$$

Therefore, the function $\mathbf{w}_{\omega\pi}$ is a weight function for $(\mathbf{p}_\omega, \mathbf{p}_{\omega\pi})$ with respect to $\Re$.

Observe the following facts which relate distributions such as $\mathbf{p}_\omega$ to $\mathbf{p}^A_{i+1}$, and $\mathbf{p}_{\omega\pi}$ to $\mathbf{p}^B_{i+1}$. For the path $\omega' \in \mathit{Path}^A_{i+1}(s_1)$ which is of the form $\omega' = \omega \xrightarrow{\sigma,p} s$, we have

$$\begin{aligned}
\mathbf{p}^A_{i+1}(\omega') &= \mathit{Prob}^A_{\mathit{fin}}(\omega') && \text{(by Definition 9.4.3)}\\
&= \mathit{Prob}^A_{\mathit{fin}}(\omega).A(\omega)(\sigma,p).p(s) && \text{(by Definition 3.2.2)}\\
&= \mathit{Prob}^A_{\mathit{fin}}(\omega).p(s) && \text{(by } A(\omega)(\sigma,p)=1)\\
&= \mathbf{p}^A_i(\omega).p(s) && \text{(by Definition 9.4.3)}\\
&= \mathbf{p}^A_i(\omega).\mathbf{p}_\omega(\omega') && \text{(by definition of } \mathbf{p}_\omega).
\end{aligned}$$

Therefore

$$\mathbf{p}^A_{i+1}(\omega') \;=\; \mathbf{p}^A_i(\omega).\mathbf{p}_\omega(\omega') \;. \tag{A.1}$$

Similarly, for the path $\pi' \in \mathit{Path}^{B_i}_{i+1}(s_2)$ which is of the form $\pi' = \pi \xrightarrow{\sigma,p'} s$, we have

$$\begin{aligned}
\mathbf{p}^B_{i+1}(\pi') &= \mathit{Prob}^{B_i}_{\mathit{fin}}(\pi') && \text{(by Definition 9.4.3)}\\
&= \mathit{Prob}^{B_{i-1}}_{\mathit{fin}}(\pi).B_i(\pi)(\sigma,p').p'(s) && \text{(by Definition 3.2.2)}\\
&= \mathbf{p}^B_i(\pi).B_i(\pi)(\sigma,p').p'(s) && \text{(by Definition 9.4.3)}\\
&= \mathbf{p}^B_i(\pi).\sum_{\omega\in \mathit{Path}^A_{\sigma,p'}(s_1)} \mathsf{Random}_i(\omega,\pi).p'(s) && \text{(by definition of } B_i(\pi)(\sigma,p'))\\
&= \mathbf{p}^B_i(\pi).\sum_{\omega\in \mathit{Path}^A_{\sigma,p'}(s_1)} \frac{\mathbf{w}_i(\omega,\pi)}{\mathbf{p}^B_i(\pi)}.p'(s) && \text{(by definition of } \mathsf{Random}_i)\\
&= \sum_{\omega\in \mathit{Path}^A_{\sigma,p'}(s_1)} \mathbf{w}_i(\omega,\pi).p'(s)\\
&= \sum_{\omega\in \mathit{Path}^A_{\sigma,p'}(s_1)} \mathbf{w}_i(\omega,\pi).\mathbf{p}_{\omega\pi}(\pi') && \text{(by definition of } \mathbf{p}_{\omega\pi}).
\end{aligned}$$

Although we may have $\mathbf{w}_i(\omega,\pi) > 0$ for paths $\omega \in \mathit{Path}^A_i(s_1) \setminus \mathit{Path}^A_{\sigma,p'}(s_1)$, we must have $\mathbf{p}_{\omega\pi}(\pi') = 0$ for such paths, from the definition of $\mathbf{p}_{\omega\pi}$ and the fact that $\pi' = \pi \xrightarrow{\sigma,p'} s$. Therefore

$$\sum_{\omega\in \mathit{Path}^A_{\sigma,p'}(s_1)} \mathbf{w}_i(\omega,\pi).\mathbf{p}_{\omega\pi}(\pi') \;=\; \sum_{\omega\in \mathit{Path}^A_i(s_1)} \mathbf{w}_i(\omega,\pi).\mathbf{p}_{\omega\pi}(\pi') \;,$$

and we can write

$$\mathbf{p}^B_{i+1}(\pi') \;=\; \sum_{\omega\in \mathit{Path}^A_i(s_1)} \mathbf{w}_i(\omega,\pi).\mathbf{p}_{\omega\pi}(\pi') \;. \tag{A.2}$$

We proceed to define the function $\mathbf{w}_{i+1} : \mathit{Path}^A_{i+1}(s_1) \times \mathit{Path}^{B_i}_{i+1}(s_2) \to [0,1]$. For the path $\omega' \in \mathit{Path}^A_{i+1}(s_1)$, $\pi' \in \mathit{Path}^{B_i}_{i+1}(s_2)$, for which the ith prefixes are denoted by $(\omega')^{(i)} = \omega$ and $(\pi')^{(i)} = \pi$, then we simply let

$$\mathbf{w}_{i+1}(\omega',\pi') = \mathbf{w}_i(\omega,\pi).\mathbf{w}_{\omega\pi}(\omega',\pi') \;.$$

We now show that $\mathbf{w}_{i+1}$ is a weight function for $(\mathbf{p}^A_{i+1}, \mathbf{p}^B_{i+1})$ with respect to $\Re$. Our first task is to show that, for the paths $\omega' \in \mathit{Path}^A_{i+1}(s_1)$, $\pi' \in \mathit{Path}^{B_i}_{i+1}(s_2)$, if $\mathbf{w}_{i+1}(\omega', \pi') > 0$, then $\omega' \Re \pi'$. We proceed through the conditions of the definition of $\Re$ (Definition A.2.1). Clearly, $|\omega'| = |\pi'| = i + 1$, which satisfies condition (1) in the definition of $\Re$. Furthermore, from $\mathbf{w}_{i+1}(\omega', \pi') > 0$, we have $\mathbf{w}_i(\omega, \pi) > 0$, which implies that $\omega \Re \pi$, thus satisfying condition (3). Finally, if $\mathbf{w}_{i+1}(\omega', \pi') > 0$ then it must be the case that $\mathbf{w}_{\omega\pi}(\omega', \pi') > 0$, which implies that $w(\mathit{last}(\omega'), \mathit{last}(\pi')) > 0$, in turn implying that $\mathit{last}(\omega') \mathcal{R} \mathit{last}(\pi')$, satisfying condition (2). Therefore, all of the conditions in the definition of $\Re$ are satisfied for ω' and π', and $\mathbf{w}_{i+1}(\omega', \pi') > 0$ implies $\omega' \Re \pi'$.

Our next task is to show that, for every path $\omega' \in \mathit{Path}^A_{i+1}(s_2)$, we have

$$\sum_{\pi' \in \mathit{Path}^{B_i}_{i+1}(s_2)} \mathbf{w}_{i+1}(\omega', \pi') = \mathbf{p}^A_{i+1}(\omega') \; .$$

Observe that the set of sets of paths

$$\{\{\pi' \in \mathit{Path}^{B_i}_{i+1}(s_2) \mid (\pi')^{(i)} = \pi\} \mid \pi \in \mathit{Path}^{B_{i-1}}_i(s_1)\}$$

is a $\emptyset$-inclusive partition of $\mathit{Path}^{B_i}_{i+1}(s_1)$. Then, from this fact and Equation 3.1, writing ω for $(\omega')^{(i)}$ as usual, we have

$$\begin{aligned}
&\sum_{\pi' \in \mathit{Path}^{B_i}_{i+1}(s_2)} \mathbf{w}_{i+1}(\omega', \pi') \\
= &\sum_{\pi \in \mathit{Path}^{B_{i-1}}_i(s_2)} \; \sum_{\substack{\pi' \in \mathit{Path}^{B_i}_{i+1}(s_2) \\ \& \ (\pi')^{(i)} = \pi}} \mathbf{w}_{i+1}(\omega', \pi') \\
= &\sum_{\pi \in \mathit{Path}^{B_{i-1}}_i(s_2)} \; \sum_{\substack{\pi' \in \mathit{Path}^{B_i}_{i+1}(s_2) \\ \& \ (\pi')^{(i)} = \pi}} \mathbf{w}_i(\omega, \pi).\mathbf{w}_{\omega\pi}(\omega', \pi') \quad \text{(by definition of } \mathbf{w}_{i+1}) \\
= &\sum_{\pi \in \mathit{Path}^{B_{i-1}}_i(s_2)} \mathbf{w}_i(\omega, \pi). \sum_{\substack{\pi' \in \mathit{Path}^{B_i}_{i+1}(s_2) \\ \& \ (\pi')^{(i)} = \pi}} \mathbf{w}_{\omega\pi}(\omega', \pi') \\
= &\sum_{\pi \in \mathit{Path}^{B_{i-1}}_i(s_2)} \mathbf{w}_i(\omega, \pi).\mathbf{p}_\omega(\omega') \qquad \text{(by Definition 3.2.12)} \\
= \;&\mathbf{p}^A_i(\omega).\mathbf{p}_\omega(\omega') \qquad \text{(by Definition 3.2.12)} \\
= \;&\mathbf{p}^A_{i+1}(\omega') \qquad \text{(by Equation A.1).}
\end{aligned}$$

The final task in proving that $\mathbf{w}_{i+1}$ is a weight function for $(\mathbf{p}^A_{i+1}, \mathbf{p}^B_{i+1})$ with respect to $\Re$ involves showing that, for every path $\pi' \in \mathit{Path}^{B_i}_{i+1}(s_1)$, we have

$$\sum_{\omega' \in \mathit{Path}^A_{i+1}(s_1)} \mathbf{w}_{i+1}(\omega', \pi') = \mathbf{p}^B_{i+1}(\pi') \; .$$

Clearly, the set of sets of paths given by

$$\{\{\omega' \in \mathit{Path}^A_{i+1}(s_1) \mid (\omega')^{(i)} = \omega\} \mid \omega \in \mathit{Path}^A_i(s_1)\}$$

is a $\emptyset$-inclusive partition of $\mathit{Path}^A_{i+1}(s_1)$. Thus, from this fact and Equation 3.1, we have

$$\begin{aligned}
&\sum_{\omega' \in \mathit{Path}^A_{i+1}(s_1)} \mathbf{w}_{i+1}(\omega', \pi') \\
= &\sum_{\omega \in \mathit{Path}^A_i(s_1)} \sum_{\substack{\omega' \in \mathit{Path}^A_{i+1}(s_1) \\ \& \ (\omega')^{(i)} = \omega}} \mathbf{w}_{i+1}(\omega', \pi') \\
= &\sum_{\omega \in \mathit{Path}^A_i(s_1)} \sum_{\substack{\omega' \in \mathit{Path}^A_{i+1}(s_1) \\ \& \ (\omega')^{(i)} = \omega}} \mathbf{w}_i(\omega, \pi).\mathbf{w}_{\omega\pi}(\omega', \pi') \quad \text{(by definition of } \mathbf{w}_{i+1}\text{)} \\
= &\sum_{\omega \in \mathit{Path}^A_i(s_1)} \mathbf{w}_i(\omega, \pi). \sum_{\substack{\omega' \in \mathit{Path}^A_{i+1}(s_1) \\ \& \ (\omega')^{(i)} = \omega}} \mathbf{w}_{\omega\pi}(\omega', \pi') \\
= &\sum_{\omega \in \mathit{Path}^A_i(s_1)} \mathbf{w}_i(\omega, \pi).\mathbf{p}_{\omega\pi}(\pi') \quad \text{(by Definition 3.2.12)} \\
= &\ \mathbf{p}^B_{i+1}(\pi') \quad \text{(by Equation A.2).}
\end{aligned}$$

Therefore, $\mathbf{w}_{i+1}$ is a weight function for $(\mathbf{p}^A_{i+1}, \mathbf{p}^B_{i+1})$ with respect to $\Re$. This concludes the proof of Proposition A.2.2. □

A.3 Properties of the constructed adversaries

In this section, we show that the adversaries A and B constructed in Section A.2 are such that the probability of satisfying an until path formula $\Phi_1 \mathcal{U} \Phi_2$, and a global path formula $\Box\Phi$, in PMC^A, is greater than or equal to the corresponding probability in PMC^B, in the case of simulation, and the probability of satisfying $\Phi_1 \mathcal{U} \Phi_2$ and $\Box\Phi$ in PMC^A and PMC^B is equal in the case of bisimulation, where PMC^A and PMC^B are the pointed Markov chains of A and s_1, and B and s_2, respectively. We proceed by introducing some notation, then deal with simulation and bisimulation in turn.

To facilitate the proof of the following proposition, we introduce two families of formulae, denoted by $\langle \Phi_1 \mathcal{U}_i \Phi_2 \rangle_{i \in \mathbb{N}}$ and $\langle \Box_i \Phi_1 \rangle_{i \in \mathbb{N}}$. For a particular $i \in \mathbb{N}$, the formula $\Phi_1 \mathcal{U}_i \Phi_2$ is taken to mean, "Φ_2 holds in the state immediately after the ith transition along the path, and Φ_1 holds in all states until the ith transition", and $\Box_i \Phi_1$ is taken to mean, "Φ_1 holds in the source and target states of the first ith transitions of the path".

Definition A.3.1 *Given $i \in \mathbb{N}$, a path $\omega \in Path_{ful}$ of a Markov chain* MC*, and the satisfaction relation $\models_{\mathcal{A}}$ defined in Definition 3.3.4, the satisfaction relations for $\Phi_1 \mathcal{U}_i \Phi_2$ and $\Box_i \Phi_1$ are defined as follows:*

$$\begin{aligned} \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2 &\Leftrightarrow \textit{there exists } 0 \leq j \leq i \textit{ such that } \omega(j) \models_{\mathcal{A}} \Phi_2, \\ &\quad\ \textit{and } \omega(k) \models_{\mathcal{A}} \Phi_1 \textit{ for all } 0 \leq k < j \\ \omega \models_{\mathcal{A}} \Box_i \Phi_1 &\Leftrightarrow \omega(j) \models_{\mathcal{A}} \Phi_1 \textit{ for all } 0 \leq j \leq i\ . \end{aligned}$$

A.3.1 Simulation and ∀PBTL

We state formally and prove the required proposition of this section. Throughout the proof, the following hypothesis will be assumed: given a set of states S, if $s, s' \in S$ are such that $s \preceq s'$, then $s' \models_{\mathcal{A}} \Phi$ implies $s \models_{\mathcal{A}} \Phi$ for any ∀PBTL formula Φ. This assumption is simply the induction hypothesis of Theorem 3.3.5. To ease notation, for a path formula φ, if a set of infinite paths $\Omega \subseteq Path_{ful}$ of a Markov chain is such that we have $\omega \models_{\mathcal{A}} \varphi$ for all paths $\omega \in \Omega$, then we write $\Omega \models_{\mathcal{A}} \varphi$.

Proposition A.3.2 *For a given concurrent probabilistic system $\mathcal{S}$, the states $s_1, s_2 \in S$ of $\mathcal{S}$ which are such that $s_1 \preceq s_2$, and a deterministic adversary A of $\mathcal{S}$, let B be a randomised adversary constructed from A using the technique of Section A.2.2. Then, for the ∀PBTL formulae Φ_1, Φ_2, we have:*

$$\begin{aligned} Prob^A(\{\omega \in Path^A_{ful}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) &\geq Prob^B(\{\pi \in Path^B_{ful}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \\ Prob^A(\{\omega \in Path^A_{ful}(s_1) \mid \omega \models_{\mathcal{A}} \Box \Phi_1\}) &\geq Prob^B(\{\pi \in Path^B_{ful}(s_2) \mid \pi \models_{\mathcal{A}} \Box \Phi_1\}) \end{aligned}$$

Proof. Let $\Re_{\preceq} \subseteq Path_{fin} \times Path_{fin}$ be the relation defined according to Definition A.2.1, and for which $\mathcal{R} = \preceq$. We proceed inductively along the length of the paths of PMC^A and PMC^B, considering the appropriate path formulae of the form $\Phi_1 \mathcal{U}_i \Phi_2$, $\Box_i \Phi_1$ as we do so. That is, for each $i \in \mathbb{N}$, we show that the probability over the set of paths of PMC^A of length i that satisfy $\Phi_1 \mathcal{U}_i \Phi_2$, is at least that of the probability over the set of paths of PMC^B that satisfy $\Phi_1 \mathcal{U}_i \Phi_2$. Formally, for each $i \in \mathbb{N}$, we have:

$$\begin{aligned} &Prob^A(\{\omega \in Path^A_{ful}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2\}) \\ \geq\ &Prob^B(\{\pi \in Path^B_{ful}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2\})\ , \end{aligned}$$

and similarly with $\Box_i \Phi_1$. Formally, The induction hypothesis takes the following form.

Induction hypothesis. For each $i \in \mathbb{N}$, and each $\omega \in Path^A_i(s_1)$, $\pi \in Path^B_i(s_2)$, if $\omega \Re_{\preceq} \pi$, then:

- $\pi{\uparrow} \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2$ implies $\omega{\uparrow} \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2$, and

- $\pi{\uparrow}\models_{\mathcal{A}} \Box_i\Phi_1$ implies $\omega{\uparrow}\models_{\mathcal{A}} \Box_i\Phi_1$.

Base. Here we consider the initial state s_1, s_2 of all of the paths of the pointed Markov chains PMC^A, PMC^B respectively, and the path formulae $\Phi_1\mathcal{U}_0\Phi_2$ and $\Box_0\Phi_1$. Observe that, from the satisfaction relation of $\Phi_1\mathcal{U}_0\Phi_2$ (Definition A.3.1), a path $\omega \in \mathit{Path}^A_{\mathit{ful}}(s_1)$ is such that $\omega \models_{\mathcal{A}} \Phi_1\mathcal{U}_0\Phi_2$ if and only if $s_1 \models_{\mathcal{A}} \Phi_2$. Similarly, for a path $\pi \in \mathit{Path}^B_{\mathit{ful}}(s_2)$, we have $\omega \models_{\mathcal{A}} \Phi_1\mathcal{U}_0\Phi_2$ if and only if $s_2 \models_{\mathcal{A}} \Phi_2$.

Observe that, when the states s_1 and s_2 are viewed as paths, they are in the relation $\Re_{\preceq}$; that is $s_1\Re_{\preceq}s_2$. Now, by the induction hypothesis of Theorem 3.3.5, the fact that $s_1 \preceq s_2$ implies that, if $s_2 \models_{\mathcal{A}} \Phi$ then $s_1 \models_{\mathcal{A}} \Phi$ for any $\forall$PBTL formula Φ. Therefore, if $s_2 \models_{\mathcal{A}} \Phi_2$ (which is equivalent to $s_2 \models_{\mathcal{A}} \Phi_1\mathcal{U}_0\Phi_2$) then $s_1 \models_{\mathcal{A}} \Phi_2$ (which is equivalent to $s_1 \models_{\mathcal{A}} \Phi_1\mathcal{U}_0\Phi_2$), which satisfies the first part of the induction hypothesis. Hence, if $\mathit{Prob}^B(\{\pi \in \mathit{Path}^B_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1\mathcal{U}_0\Phi_2\}) = 1$ then $\mathit{Prob}^A(\{\omega \in \mathit{Path}^A_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1\mathcal{U}_0\Phi_2\}) = 1$. Therefore:

$$\begin{aligned}&\mathit{Prob}^A(\{\omega \in \mathit{Path}^A_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1\mathcal{U}_0\Phi_2\})\\ \geq\ & \mathit{Prob}^B(\{\pi \in \mathit{Path}^B_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1\mathcal{U}_0\Phi_2\})\ .\end{aligned}$$

Similarly, if $s_2 \models_{\mathcal{A}} \Phi_1$ then $s_1 \models_{\mathcal{A}} \Phi_1$, and therefore if $s_2{\uparrow}\models_{\mathcal{A}} \Box_0\Phi_1$ then $s_1{\uparrow}\models_{\mathcal{A}} \Box_0\Phi_1$. Hence, both parts of the induction hypothesis hold.

Induction. For $i \geq 1$, we consider the path formulae $\Phi_1\mathcal{U}_i\Phi_2$ and $\Box_i\Phi_1$. To determine the probabilities of paths of PMC^A and PMC^B satisfying $\Phi_1\mathcal{U}_i\Phi_2$, it suffices to consider finite prefixes of paths in $\mathit{Path}^A_{\mathit{ful}}(s_1)$ and $\mathit{Path}^B_{\mathit{ful}}(s_2)$ of length i (we then reason about the basic cylinders of such prefixes). Recall that we defined the distributions $\mathbf{p}^A_i$ and $\mathbf{p}^B_i$ such that, for any $\omega \in \mathit{Path}^A_i(s_1)$, we have $\mathbf{p}^A_i(\omega) = \mathit{Prob}^A_{\mathit{fin}}(\omega)$, and, similarly, for any $\pi \in \mathit{Path}^B_i(s_2)$, we have $\mathbf{p}^B_i(\pi) = \mathit{Prob}^B_{\mathit{fin}}(\pi)$. Furthermore, by Proposition A.2.2, we also have $\mathbf{p}^A_i\Re_{\preceq}\mathbf{p}^B_i$.

Let $\Omega_{\mathcal{U}} \subseteq \mathit{Path}^A_i(s_1)$ be the set of paths of length i of A that are prefixes of paths that satisfy $\Phi_1\mathcal{U}_i\Phi_2$; that is:

$$\Omega_{\mathcal{U}} = \{\omega \in \mathit{Path}^A_i(s_1) \mid \omega{\uparrow}\models_{\mathcal{A}} \Phi_1\mathcal{U}_i\Phi_2\}\ .$$

Similarly, let $\Pi_{\mathcal{U}} \subseteq \mathit{Path}^B_i(s_2)$ be the set of paths of length i of B such that:

$$\Pi_{\mathcal{U}} = \{\pi \in \mathit{Path}^B_i(s_2) \mid \pi{\uparrow}\models_{\mathcal{A}} \Phi_1\mathcal{U}_i\Phi_2\}\ .$$

Now consider two paths $\omega \in \mathit{Path}^A_i(s_1)$, $\pi \in \mathit{Path}^B_i(s_2)$ which are such that $\omega\Re_{\preceq}\pi$. The $\Re_{\preceq}$ relation between ω and π implies the following: for all $0 \leq j < i$, we have $\omega^{(j)}\Re_{\preceq}\pi^{(j)}$, and $\mathit{last}(\omega) \preceq \mathit{last}(\pi)$. By induction, if there exists $0 \leq j < i$ such that $\pi^{(j)}{\uparrow}\models_{\mathcal{A}} \Phi_1\mathcal{U}_j\Phi_2$, it must be the case that $\omega^{(j)}{\uparrow}\models_{\mathcal{A}} \Phi_1\mathcal{U}_j\Phi_2$. Also by induction, if $\pi^{(i-1)}{\uparrow}\models_{\mathcal{A}} \Box_{i-1}\Phi_1$ then $\omega^{(i-1)}{\uparrow}\models_{\mathcal{A}} \Box_{i-1}\Phi_1$. Furthermore, from $\mathit{last}(\omega) \preceq \mathit{last}(\pi)$ and the induction hypothesis of Theorem 3.3.5, we have $\mathit{last}(\pi) \models_{\mathcal{A}} \Phi_2$ implies $\mathit{last}(\omega) \models_{\mathcal{A}} \Phi_2$.

Combining the last two facts, then if $\pi\!\uparrow \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2$, we conclude that it must be the case that $\omega\!\uparrow \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2$, thus preserving the first part of the induction hypothesis. Hence, the relation $\Re_{\preceq}$ is such that, if $\omega \Re_{\preceq} \pi$, then $\pi \in \Pi_{\mathcal{U}}$ implies $\omega \in \Omega_{\mathcal{U}}$. Therefore, from Lemma 3.2.13, we have:

$$\sum_{\omega \in \Omega_{\mathcal{U}}} \mathbf{p}_i^A(\omega) \geq \sum_{\pi \in \Pi_{\mathcal{U}}} \mathbf{p}_i^B(\pi) \ . \tag{A.3}$$

Now, from the fact that

$$\begin{aligned} \bigcup_{\omega \in \Omega_{\mathcal{U}}} \omega\!\uparrow &= \{\omega' \in \mathit{Path}_{\mathit{ful}}^A(s_1) \mid \omega' \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2\} \\ \bigcup_{\pi \in \Pi_{\mathcal{U}}} \pi\!\uparrow &= \{\pi' \in \mathit{Path}_{\mathit{ful}}^B(s_2) \mid \pi' \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2\} \ , \end{aligned}$$

Lemma 3.2.4 and the definition of path distributions (Definition 9.4.3), it follows that:

$$\begin{aligned} \mathit{Prob}^A(\{\omega \in \mathit{Path}_{\mathit{ful}}^A(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2\}) &= \sum_{\omega \in \Omega_{\mathcal{U}}} \mathbf{p}_i^A(\omega) \\ \mathit{Prob}^B(\{\pi \in \mathit{Path}_{\mathit{ful}}^B(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2\}) &= \sum_{\pi \in \Pi_{\mathcal{U}}} \mathbf{p}_i^B(\pi) \ . \end{aligned}$$

Therefore, by Equation A.3, we have:

$$\begin{aligned} &\mathit{Prob}^A(\{\omega \in \mathit{Path}_{\mathit{ful}}^A(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2\}) \\ \geq\ &\mathit{Prob}^B(\{\pi \in \mathit{Path}_{\mathit{ful}}^B(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2\}) \ . \end{aligned}$$

We turn our attention to the path formula $\Box_i \Phi_1$ and the preservation of the second part of the induction hypothesis. Recall that our aim is to show that if $\omega \Re_{\preceq} \pi$ then $\pi\!\uparrow \models_{\mathcal{A}} \Box_i \Phi_1$ implies $\omega\!\uparrow \models_{\mathcal{A}} \Box_i \Phi_1$ for each $\omega \in \mathit{Path}_i^A(s_1)$, $\pi \in \mathit{Path}_i^B(s_2)$. From $\omega \Re_{\preceq} \pi$, we have $\omega^{(i-1)} \Re_{\preceq} \pi^{(i-1)}$, and $\mathit{last}(\omega) \preceq \mathit{last}(\pi)$; therefore, by induction, if $\pi^{(i-1)}\!\uparrow \models_{\mathcal{A}} \Box_{i-1} \Phi_1$ then $\omega^{(i-1)}\!\uparrow \models_{\mathcal{A}} \Box_{i-1} \Phi_1$ (corresponding to the case in which, for all $0 \leq j \leq (i-1)$, we have $\omega(j) \models_{\mathcal{A}} \Phi_1$), and by the induction hypothesis of Theorem 3.3.5, we have $\mathit{last}(\pi) \models_{\mathcal{A}} \Phi_1$ implies $\mathit{last}(\omega) \models_{\mathcal{A}} \Phi_1$ (corresponding to the case in which, we have $\omega(i) \models_{\mathcal{A}} \Phi_1$). Therefore, if $\pi\!\uparrow \models_{\mathcal{A}} \Box_i \Phi_1$ then $\omega\!\uparrow \models_{\mathcal{A}} \Box_i \Phi_1$, thus preserving the second part of the induction hypothesis. Furthermore, using the same reasoning as that used for $\Phi_1 \mathcal{U}_i \Phi_2$, we have:

$$\mathit{Prob}^A(\{\omega \in \mathit{Path}_{\mathit{ful}}^A(s_1) \mid \omega \models_{\mathcal{A}} \Box_i \Phi_1\}) \geq \mathit{Prob}^B(\{\pi \in \mathit{Path}_{\mathit{ful}}^B(s_2) \mid \pi \models_{\mathcal{A}} \Box_i \Phi_1\}) \ .$$

Now observe that, for the set of infinite paths $\mathit{Path}_{\mathit{ful}}$ of any Markov chain MC, from the definitions of the satisfaction relation of $\Phi_1 \mathcal{U} \Phi_2$ and $\Phi_1 \mathcal{U}_i \Phi_2$, we have

$$\{\omega \in \mathit{Path}_{\mathit{ful}} \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\} = \bigcup_{i \in \mathbb{N}} \{\omega \in \mathit{Path}_{\mathit{ful}} \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2\} \ .$$

As we have shown that the inequality in

$$\begin{array}{ll} & \mathit{Prob}^A(\{\omega \in \mathit{Path}^A_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2\}) \\ \geq & \mathit{Prob}^B(\{\pi \in \mathit{Path}^B_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2\}) \ . \end{array}$$

holds for all $i \in \mathbb{N}$, we conclude that

$$\mathit{Prob}^A(\{\omega \in \mathit{Path}^A_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \geq \mathit{Prob}^B(\{\pi \in \mathit{Path}^B_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \ .$$

Similarly, we have:

$$\mathit{Prob}^A(\{\omega \in \mathit{Path}^A_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Box \Phi_1\}) \geq \mathit{Prob}^B(\{\pi \in \mathit{Path}^B_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Box \Phi_1\}) \ .$$

□

We now recall, and prove, Theorem 3.3.5.

Theorem 3.3.5 *Let $\mathcal{S}$ be a concurrent probabilistic system, let $\mathcal{A}$ be a set of adversaries of $\mathcal{S}$, and let Φ be a $\forall$PBTL formula. If $s_1 \preceq s_2$, for $s_1, s_2 \in S$, then $s_2 \models_{\mathcal{A}} \Phi$ implies $s_1 \models_{\mathcal{A}} \Phi$.*

Proof. We proceed by induction on the length of the subformulae of Φ. The cases for atomic propositions and boolean connectives follow in a straightforward manner, and therefore we concentrate on the case of a formula of the form $[\Phi_1 \forall \mathcal{U} \Phi_2]_{\sqsupseteq \lambda}$, where Φ_1, Φ_2 are $\forall$PBTL formulae, $\sqsupseteq \in \{\geq, >\}$, and $\lambda \in [0,1]$, with the case for $[\forall \Box \Phi]_{\sqsupseteq \lambda}$ following in a similar manner. Assume by induction that we have evaluated the validity of Φ_1 and Φ_2 for all of the states of $\mathcal{S}$.

Consider the sets *Adv* and *Radv* of all deterministic and randomised adversaries, respectively, of $\mathcal{S}$. Let $A_{\min} \in \mathit{Adv}$ be the deterministic adversary such that

$$\begin{array}{ll} & \mathit{Prob}^{A_{\min}}(\{\omega \in \mathit{Path}^A_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \\ = & \inf_{A \in \mathit{Adv}} \mathit{Prob}^A(\{\omega \in \mathit{Path}^A_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \ . \end{array}$$

That is, the deterministic adversary $A_{\min}$ assigns the minimal probability over all deterministic adversaries to the satisfaction of $\Phi_1 \mathcal{U} \Phi_2$ for paths starting from s_1. Let $B_{A_{\min}} \in \mathit{Radv}$ be the randomised adversary constructed from $A_{\min}$ using the construction technique of Section A.2.2. From Proposition A.3.2, we have

$$\begin{array}{ll} & \mathit{Prob}^{A_{\min}}(\{\omega \in \mathit{Path}^A_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \\ \geq & \mathit{Prob}^{B_{A_{\min}}}(\{\omega \in \mathit{Path}^B_{\mathit{ful}}(s_2) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \ . \end{array} \tag{A.4}$$

Now let $B_{\min} \in \mathit{Radv}$ be the randomised adversary such that

$$\begin{array}{ll} & \mathit{Prob}^{B_{\min}}(\{\pi \in \mathit{Path}^B_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \\ = & \inf_{B \in \mathit{Radv}} \mathit{Prob}^B(\{\pi \in \mathit{Path}^B_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \ . \end{array}$$

Naturally, the randomised adversary $B_{\min}$ assigns the minimal probability over all randomised adversaries to the satisfaction of $\Phi_1 \mathcal{U} \Phi_2$ for paths starting from s_2. Note that, by the definition of $B_{\min}$, it must be the case that

$$\begin{array}{ll} & \mathit{Prob}^{B_{A_{\min}}}(\{\omega \in \mathit{Path}^{A}_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \\ \geq & \mathit{Prob}^{B_{\min}}(\{\omega \in \mathit{Path}^{B}_{\mathit{ful}}(s_2) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \ . \end{array} \tag{A.5}$$

By Corollary 20 of [BdA95], the adversary $B_{\min}$ is a deterministic adversary; that is, we have $B \in \mathit{Adv}$. Combining Equation A.4 and Equation A.5, we have

$$\begin{array}{ll} & \mathit{Prob}^{A_{\min}}(\{\omega \in \mathit{Path}^{A}_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \\ \geq & \mathit{Prob}^{B_{\min}}(\{\pi \in \mathit{Path}^{B}_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \ . \end{array}$$

Now recall from Definition 3.3.2 that, for any state $s \in S$, we have

$$\begin{array}{lll} s \models_{\mathcal{A}} [\Phi_1 \forall \mathcal{U}\, \Phi_2]_{\sqsupseteq \lambda} & \Leftrightarrow & \mathit{Prob}^{A}(\{\omega \in \mathit{Path}^{A}_{\mathit{ful}}(s) \ \mid \ \omega \models_{\mathcal{A}} \Phi_1 \, \mathcal{U} \, \Phi_2\}) \sqsupseteq \lambda \\ & & \text{for all (deterministic) adversaries } A \in \mathcal{A} \ . \end{array}$$

If $\mathit{Prob}^{B_{\min}}(\{\pi \in \mathit{Path}^{B}_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \sqsupseteq \lambda$, then $\mathit{Prob}^{A_{\min}}(\{\omega \in \mathit{Path}^{A}_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \sqsupseteq \lambda$. By the definitions of $A_{\min}, B_{\min}$ (in that they assign the minimal probability to the satisfaction of $\Phi_1 \mathcal{U} \Phi_2$), then $s_2 \models_{\mathcal{A}} [\Phi_1 \forall \mathcal{U} \Phi_2]_{\sqsupseteq \lambda}$ implies $s_1 \models_{\mathcal{A}} [\Phi_1 \forall \mathcal{U} \Phi_2]_{\sqsupseteq \lambda}$. □

A.3.2 Bisimulation and PBTL

Consider the sets of paths of the adversaries A and B from the states s_1 and s_2 respectively, where s_1 and s_2 are bisimilar. The following proposition states that the probability of satisfying the until formula $\Phi_1 \mathcal{U} \Phi_2$, where Φ_1 and Φ_2 are PBTL formulae, is the same for these two sets of paths.

Proposition A.3.3 *For a given concurrent probabilistic system $\mathcal{S}$, the states $s_1, s_2 \in S$ of $\mathcal{S}$ which are such that $s_1 \simeq s_2$, and a deterministic adversary A of $\mathcal{S}$, let B be a randomised adversary constructed from A using the technique of Section A.2.2. Then, for the PBTL formulae Φ_1, Φ_2, we have:*

$$\mathit{Prob}^{A}(\{\omega \in \mathit{Path}^{A}_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) = \mathit{Prob}^{B}(\{\pi \in \mathit{Path}^{B}_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \ .$$

Proof. Let $\Re_{\simeq} \subseteq \mathit{Path}_{\mathit{fin}} \times \mathit{Path}_{\mathit{fin}}$ be the relation defined according to Definition A.2.1, and for which $\mathcal{R} = \simeq$. The proof is similar to that of Proposition A.3.2, and therefore we only sketch many details. As in the proof of Proposition A.3.2, we proceed inductively along the length of the paths of PMC^A and PMC^B, considering the appropriate path formulae $\Phi_1 \mathcal{U}_i \Phi_2$, $\Box_i \Phi_1$ at each step. Then, for each $i \geq 1$, we have:

$$\begin{array}{ll} & \mathit{Prob}^{A}(\{\omega \in \mathit{Path}^{A}_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2\}) \\ = & \mathit{Prob}^{B}(\{\pi \in \mathit{Path}^{B}_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2\}) \ . \end{array}$$

Formally, the induction hypothesis takes the following form.

Induction hypothesis. For each $i \in \mathbb{N}$, and each $\omega \in \mathit{Path}_i^A(s_1)$, $\pi \in \mathit{Path}_i^B(s_2)$, if $\omega \Re_{\simeq} \pi$, then:

- $\omega{\uparrow} \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2$ if and only if $\pi{\uparrow} \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2$, and
- $\omega{\uparrow} \models_{\mathcal{A}} \Box_i \Phi_1$ if and only if $\pi{\uparrow} \models_{\mathcal{A}} \Box_i \Phi_1$.

Base. Here we consider the initial state s_1, s_2 of all of the paths of the pointed Markov chains PMC^A, PMC^B respectively, and the path formulae $\Phi_1 \mathcal{U}_0 \Phi_2$ and $\Box_0 \Phi_1$. Observe that, from the satisfaction relation of $\Phi_1 \mathcal{U}_0 \Phi_2$ (Definition A.3.1), a path $\omega \in \mathit{Path}_{\mathit{ful}}^A(s_1)$ is such that $\omega \models_{\mathcal{A}} \Phi_1 \mathcal{U}_0 \Phi_2$ if and only if $s_1 \models_{\mathcal{A}} \Phi_2$. Similarly, for a path $\pi \in \mathit{Path}_{\mathit{ful}}^B(s_2)$, we have $\pi \models_{\mathcal{A}} \Phi_1 \mathcal{U}_0 \Phi_2$ if and only if $s_2 \models_{\mathcal{A}} \Phi_2$.

Consider the view that the states s_1 and s_2 are also paths of length 0. Observe that $s_1 \Re_{\simeq} s_2$ by Definition A.2.1 and the fact that $\mathcal{R} = \simeq$. By the induction hypothesis of Theorem 3.3.6, the fact that $s_1 \simeq s_2$ implies that $s_1 \models_{\mathcal{A}} \Phi$ if and only if $s_2 \models_{\mathcal{A}} \Phi$ for any PBTL formula Φ. Therefore, we have $s_1 \models_{\mathcal{A}} \Phi_2$ (which is equivalent to $s_1 \models_{\mathcal{A}} \Phi_1 \mathcal{U}_0 \Phi_2$) if and only if $s_2 \models_{\mathcal{A}} \Phi_2$ (which is equivalent to $s_2 \models_{\mathcal{A}} \Phi_1 \mathcal{U}_0 \Phi_2$), which satisfies the first part of the induction hypothesis. Hence, we have $\mathit{Prob}^A(\{\omega \in \mathit{Path}_{\mathit{ful}}^A(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U}_0 \Phi_2\}) = 1$ if and only if $\mathit{Prob}^B(\{\pi \in \mathit{Path}_{\mathit{ful}}^B(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U}_0 \Phi_2\}) = 1$, otherwise both of these probabilities are equal to 0. Therefore:

$$\begin{aligned}&\mathit{Prob}^A(\{\omega \in \mathit{Path}_{\mathit{ful}}^A(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U}_0 \Phi_2\})\\ &= \mathit{Prob}^B(\{\pi \in \mathit{Path}_{\mathit{ful}}^B(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U}_0 \Phi_2\})\ .\end{aligned}$$

Similarly, we have $s_1 \models_{\mathcal{A}} \Phi_1$ if and only if $s_2 \models_{\mathcal{A}} \Phi_1$, and therefore $s_1{\uparrow} \models_{\mathcal{A}} \Box_0 \Phi_1$ if and only if $s_2{\uparrow} \models_{\mathcal{A}} \Box_0 \Phi_1$. Hence, both parts of the induction hypothesis hold.

Induction. For $i \geq 1$, we consider the path formulae $\Phi_1 \mathcal{U}_i \Phi_2$ and $\Box_i \Phi_1$. As in the proof of Proposition A.3.2, we determine the probabilities of paths of PMC^A and PMC^B that satisfy $\Phi_1 \mathcal{U}_i \Phi_2$ by considering finite prefixes of paths in $\mathit{Path}_{\mathit{ful}}^A(s_1)$ and $\mathit{Path}_{\mathit{ful}}^B(s_2)$ of length i. Recall the definitions of the distributions $\mathbf{p}_i^A$ and $\mathbf{p}_i^B$ (Definition 9.4.3). Furthermore, by Lemma A.2.2, we also have $\mathbf{p}_i^A \Re_{\simeq} \mathbf{p}_i^B$.

Let $\Omega_{\mathcal{U}} \subseteq \mathit{Path}_i^A(s_1)$ and $\Pi_{\mathcal{U}} \subseteq \mathit{Path}_i^B(s_2)$ be the sets of paths of length i of A and B, respectively, that satisfy $\Phi_1 \mathcal{U}_i \Phi_2$ (for a formal definition, see the proof of Proposition A.3.2). Now consider two paths $\omega \in \mathit{Path}_i^A(s_1)$, $\pi \in \mathit{Path}_i^B(s_2)$ which are such that $\omega \Re_{\simeq} \pi$. By definition of Definition 3.2.16, and for all $0 \leq j < i$, we have $\omega^{(j)} \Re_{\simeq} \pi^{(j)}$, and $\mathit{last}(\omega) \simeq \mathit{last}(\pi)$. By induction, there exists $0 \leq j < i$ such that $\omega^{(j)} \models_{\mathcal{A}} \Phi_1 \mathcal{U}_j \Phi_2$ if and only if $\pi^{(j)} \models_{\mathcal{A}} \Phi_1 \mathcal{U}_j \Phi_2$. Also by induction, we have $\omega^{(i-1)}{\uparrow} \models_{\mathcal{A}} \Box_{i-1} \Phi_1$ if and only if $\pi^{(i-1)}{\uparrow} \models_{\mathcal{A}} \Box_{i-1} \Phi_1$. Furthermore, from $\mathit{last}(\omega) \simeq \mathit{last}(\pi)$ and the induction hypothesis of Theorem 3.3.6, we have $\mathit{last}(\omega) \models_{\mathcal{A}} \Phi_2$ if and only if $\mathit{last}(\pi) \models_{\mathcal{A}} \Phi_2$. Therefore, we have $\omega{\uparrow} \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2$ if and only if $\pi{\uparrow} \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2$, thus preserving the first part of the induction hypothesis. Hence, the relation $\Re_{\simeq}$ is such that, if $\omega \Re_{\simeq} \pi$, then $\omega \in \Omega_{\mathcal{U}}$ if and only if $\pi \in \Pi_{\mathcal{U}}$. Therefore, from Lemma 3.2.14, we have:

$$\sum_{\omega \in \Omega_{\mathcal{U}}} \mathbf{p}_i^A(\omega) = \sum_{\pi \in \Pi_{\mathcal{U}}} \mathbf{p}_i^B(\pi) \; .$$

Using similar reasoning as that used in the proof of Proposition A.3.2, we have:

$$\begin{aligned} & \mathit{Prob}^A(\{\omega \in \mathit{Path}^A_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2\}) \\ = \;\; & \mathit{Prob}^B(\{\pi \in \mathit{Path}^B_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U}_i \Phi_2\}) \; . \end{aligned}$$

We now deal with the second part of the preservation of the induction hypothesis, which concerns the path formula $\Box_i \Phi_1$. Recall that our aim is to show that if $\omega \Re_{\simeq} \pi$ then $\omega{\uparrow} \models_{\mathcal{A}} \Box_i \Phi_1$ if and only if $\pi{\uparrow} \models_{\mathcal{A}} \Box_i \Phi_1$ for each $\omega \in \mathit{Path}^A_i(s_1)$, $\pi \in \mathit{Path}^B_i(s_2)$. From $\omega \Re_{\simeq} \pi$, we have $\omega^{(i-1)} \Re_{\simeq} \pi^{(i-1)}$, and $\mathit{last}(\omega) \simeq \mathit{last}(\pi)$; therefore, by induction, we have $\omega^{(i-1)}{\uparrow} \models_{\mathcal{A}} \Box_{i-1} \Phi_1$ if and only if $\pi^{(i-1)}{\uparrow} \models_{\mathcal{A}} \Box_{i-1} \Phi_1$, and by the induction hypothesis of Theorem 3.3.6, we have $\mathit{last}(\omega) \models_{\mathcal{A}} \Phi_1$ if and only if $\mathit{last}(\pi) \models_{\mathcal{A}} \Phi_1$. Therefore, we have $\omega{\uparrow} \models_{\mathcal{A}} \Box_i \Phi_1$ if and only if $\pi{\uparrow} \models_{\mathcal{A}} \Box_i \Phi_1$, thus preserving the second part of the induction hypothesis.

Again, using similar intuition to that used in the proof of Proposition A.3.2, we conclude that

$$\mathit{Prob}^A(\{\omega \in \mathit{Path}^A_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) = \mathit{Prob}^B(\{\pi \in \mathit{Path}^B_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \; .$$

□

We are now in a position to prove Theorem 3.3.6.

Theorem 3.3.6 *Let $\mathcal{S}$ be a concurrent probabilistic system, let $\mathcal{A}$ be a set of adversaries of $\mathcal{S}$, and let Φ be a PBTL formula. If $s_1 \simeq s_2$, for $s_1, s_2 \in S$, then $s_1 \models_{\mathcal{A}} \Phi$ if and only if $s_2 \models_{\mathcal{A}} \Phi$.*

Proof. Consider the sets *Adv* and *Radv* of deterministic and randomised adversaries of $\mathcal{S}$, respectively, and let $A_{\min}, B_{\min} \in \mathit{Adv}$, $B_{A_{\min}} \in \mathit{Radv}$ be the adversaries as defined in the proof of Theorem 3.3.5. Then

$$\begin{aligned} & \mathit{Prob}^{A_{\min}}(\{\omega \in \mathit{Path}^A_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \\ = \;\; & \mathit{Prob}^{B_{A_{\min}}}(\{\pi \in \mathit{Path}^B_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \quad \text{(by Proposition A.3.3)} \\ \geq \;\; & \mathit{Prob}^{B_{\min}}(\{\pi \in \mathit{Path}^B_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \quad \text{(by definition of } B_{\min}\text{).} \end{aligned}$$

However, as bisimulation is an equivalence relation, we can reverse the roles of the adversaries $A_{\min}, B_{A_{\min}}$ and $B_{\min}$. That is, for the deterministic adversary $B_{\min}$, which defines paths from the state s_2, we can construct the randomised adversary $A_{B_{\min}} \in$

Radv, which is such that

$$\begin{array}{ll} & \mathit{Prob}^{B_{\min}}(\{\pi \in \mathit{Path}^{B}_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \\ = & \mathit{Prob}^{A_{B_{\min}}}(\{\omega \in \mathit{Path}^{A}_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \quad \text{(by Proposition A.3.3)} \\ \geq & \mathit{Prob}^{A_{\min}}(\{\omega \in \mathit{Path}^{A}_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \quad \text{(by definition of } A_{\min}). \end{array}$$

Combining these two equations means that we have

$$\begin{array}{ll} & \mathit{Prob}^{A_{\min}}(\{\omega \in \mathit{Path}^{A}_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \\ = & \mathit{Prob}^{B_{\min}}(\{\pi \in \mathit{Path}^{B}_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \ . \end{array}$$

Now consider the PBTL formula $[\Phi_1 \forall \mathcal{U} \Phi_2]_{\sqsupseteq \lambda}$. The intuition involved in showing that $s_1 \models_{\mathcal{A}} [\Phi_1 \forall \mathcal{U} \Phi_2]_{\sqsupseteq \lambda}$ if and only if $s_2 \models_{\mathcal{A}} [\Phi_1 \forall \mathcal{U} \Phi_2]_{\sqsupseteq \lambda}$ is similar to that employed in the proof of Theorem 3.3.5; that is, $\mathit{Prob}^{A_{\min}}(\{\omega \in \mathit{Path}^{A}_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \sqsupseteq \lambda$ if and only if $\mathit{Prob}^{B_{\min}}(\{\pi \in \mathit{Path}^{B}_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \sqsupseteq \lambda$, and, by the definition of $A_{\min}, B_{\min}$, we then have $s_1 \models_{\mathcal{A}} [\Phi_1 \forall \mathcal{U} \Phi_2]_{\sqsupseteq \lambda}$ if and only if $s_2 \models_{\mathcal{A}} [\Phi_1 \forall \mathcal{U} \Phi_2]_{\sqsupseteq \lambda}$.

This process can be repeated also for the adversaries that are *maximal* in terms of the satisfaction of the path formula $\Phi_1 \mathcal{U} \Phi_2$. That is, given that we have defined $A_{\max} \in \mathit{Adv}$ and $B_{\max}$ by

$$\begin{array}{ll} \mathit{Prob}^{A_{\max}}(\{\omega \in \mathit{Path}^{A}_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) & \\ \qquad = \ \sup_{A \in \mathit{Adv}} \mathit{Prob}^{A}(\{\omega \in \mathit{Path}^{A}_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) & \\ \mathit{Prob}^{B_{\max}}(\{\pi \in \mathit{Path}^{B}_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) & \\ \qquad = \ \sup_{B \in \mathit{Radv}} \mathit{Prob}^{B}(\{\pi \in \mathit{Path}^{B}_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \ , & \end{array}$$

respectively, then using a method symmetric to that used above, we can show that

$$\begin{array}{ll} & \mathit{Prob}^{A_{\max}}(\{\omega \in \mathit{Path}^{A}_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \\ = & \mathit{Prob}^{B_{\max}}(\{\pi \in \mathit{Path}^{B}_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \ . \end{array}$$

Then, for the PBTL formula $[\Phi_1 \exists \mathcal{U} \Phi_2]_{\sqsupseteq \lambda}$, we have $\mathit{Prob}^{A_{\max}}(\{\omega \in \mathit{Path}^{A}_{\mathit{ful}}(s_1) \mid \omega \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \sqsupseteq \lambda$ if and only if $\mathit{Prob}^{B_{\max}}(\{\pi \in \mathit{Path}^{B}_{\mathit{ful}}(s_2) \mid \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U} \Phi_2\}) \sqsupseteq \lambda$, and, by the definition of $A_{\max}, B_{\max}$, we then have $s_1 \models_{\mathcal{A}} [\Phi_1 \exists \mathcal{U} \Phi_2]_{\sqsupseteq \lambda}$ if and only if $s_2 \models_{\mathcal{A}} [\Phi_1 \exists \mathcal{U} \Phi_2]_{\sqsupseteq \lambda}$. □

Appendix B

Steam boiler: additional information

B.1 Symbolic states generated by FRRectangular

We present the states reachable with probability strictly greater than 0 of the probabilistic hybrid automaton R_{sb}, subject to the target location Target, which (informally speaking) can be reached from any of the non-terminal locations of R_{sb} if the global clock c is greater than or equal to 25. As dictated by the algorithm **FRRectangular**, and its use of the operation post given in Definition 9.2.3, the reachable states have a natural representation as a set of symbolic states, the second component of which take the form of convex polyhedra. The presentation of these state sets will take the following form:

Code	Constraint	Targets
...	...	...
N3	$5 \leq w \leq 25 + 3t \wedge 15 \leq c \leq 20$	(F8,UF6) or (N4,UN7)
...	...	...

The code refers to both the location component of the symbolic state, which is represented as one or two letters, and a numeral to distinguish different convex polyhedra. That is, we have a different code for each generated symbolic state. Recall that the way in which locations are denoted as letters is given in Table 9.1. The polyhedron of the symbolic state is then given in the column marked "Constraint", and is represented in a syntactic form as a conjunction of linear constraints on variable values. Finally, the column labelled "Targets" gives information concerning the transitions available in this state. Entries in this column consist of tuples of symbolic state codes separated by the word "or", with the meaning that each tuple corresponds to a distribution. Therefore, a nondeterministic choice is made between the distributions, and then a probabilistic choice is made between the symbolic states given by the codes within the

Code	Constraint	Targets
F1	$15 - 2t \leq w \leq 15 \wedge 0 \leq c \leq 5$	(F2,UF1) or (N1,UN1)
F2	$10 - 2t \leq w \leq 15 \wedge 5 \leq c \leq 10$	(F3,UF2) or (N5,UN2)
F3	$10 - 2t \leq w \leq 15 \wedge 10 \leq c \leq 15$	(F4,UF3) or (N8,UN3)
F4	$10 - 2t \leq w \leq 15 \wedge 15 \leq c \leq 20$	(F5,UF4) or (N10,UN4)
F5	$10 - 2t \leq w \leq 15 \wedge 20 \leq c \leq 25$	TARGET
F6	$25 - 2t < w < 40 \wedge 15 \leq c \leq 20$	(F7,UF12)
F7	$15 - 2t < w < 40 \wedge 20 \leq c \leq 25$	TARGET
F8	$25 - 2t < w \leq 40 \wedge 20 \leq c \leq 25$	TARGET
F9	$25 - 2t < w < 40 \wedge 20 \leq c \leq 25$	TARGET
F10	$25 - 2t < w \leq 30 \wedge 15 \leq c \leq 20$	(F11,UF13)
F11	$15 - 2t < w \leq 30 \wedge 20 \leq c \leq 25$	TARGET
F12	$25 - 2t < w < 30 \wedge 15 \leq c \leq 20$	(F13,UF14)
F13	$15 - 2t < w < 30 \wedge 20 \leq c \leq 25$	TARGET
F14	$25 - 2t < w \leq 40 \wedge 20 \leq c \leq 25$	TARGET
F15	$25 - 2t < w \leq 30 \wedge 20 \leq c \leq 25$	TARGET
F16	$25 - 2t < w < 30 \wedge 20 \leq c \leq 25$	TARGET
F17	$20 - 2t < w < 40 \wedge 15 \leq c \leq 20$	(F18,UF15)
F18	$10 - 2t < w < 40 \wedge 20 \leq c \leq 25$	TARGET
F19	$15 - 2t \leq w \leq 35 \wedge 15 \leq c \leq 20$	(F20,UF16) or (N8,UN12)
F20	$10 - 2t \leq w \leq 30 \wedge 20 \leq c \leq 25$	TARGET
F21	$20 - 2t < w \leq 40 \wedge 20 \leq c \leq 25$	TARGET
F22	$15 - 2t \leq w \leq 35 \wedge 20 \leq c \leq 25$	TARGET
F23	$20 - 2t < w < 40 \wedge 20 \leq c \leq 25$	TARGET
F24	$15 - 2t < w < 40 \wedge 20 \leq c \leq 25$	TARGET

Table B.1: Symbolic states computed for the location Off.

Code	Constraint	Targets
N1	$5 \leq w < 10 + 3t \wedge 5 \leq c \leq 10$	(N2,UN5)
N2	$5 \leq w < 25 + 3t \wedge 10 \leq c \leq 15$	(F6,UF5) or (N3,UN6)
N3	$5 \leq w \leq 25 + 3t \wedge 15 \leq c \leq 20$	(F8,UF6) or (N4,UN7)
N4	$5 \leq w \leq 25 + 3t \wedge 20 \leq c \leq 25$	TARGET
N5	$0 \leq w < 10 + 3t \wedge 10 \leq c \leq 15$	(N6,UN8)
N6	$0 \leq w < 25 + 3t \wedge 15 \leq c \leq 20$	(F9,UF7) or (N7,UN9)
N7	$0 \leq w \leq 25 + 3t \wedge 20 \leq c \leq 25$	TARGET
N8	$0 \leq w < 10 + 3t \wedge 15 \leq c \leq 20$	(N9,UN10)
N9	$0 \leq w < 25 + 3t \wedge 20 \leq c \leq 25$	TARGET
N10	$0 \leq w < 10 + 3t \wedge 20 \leq c \leq 25$	TARGET
N11	$5 \leq w \leq 15 + 3t \wedge 10 \leq c \leq 15$	(F10,UF8) or (N3,UN6)
N12	$0 \leq w < 15 + 3t \wedge 10 \leq c \leq 15$	(F12,UF9) or (N13,UN11)
N13	$0 \leq w \leq 25 + 3t \wedge 15 \leq c \leq 20$	(F14,UF6) or (N7,UN9)
N14	$5 \leq w \leq 15 + 3t \wedge 15 \leq c \leq 20$	(F15,UF10) or (N4,UN7)
N15	$0 \leq w < 15 + 3t \wedge 15 \leq c \leq 20$	(F16,UF11) or (N7,UN9)
N16	$5 \leq w \leq 15 + 3t \wedge 20 \leq c \leq 25$	TARGET
N17	$0 \leq w < 15 + 3t \wedge 20 \leq c \leq 25$	TARGET
N18	$5 \leq w < 10 + 3t \wedge 20 \leq c \leq 25$	TARGET

Table B.2: Symbolic states computed for the location On.

Code	Constraint	Targets
UF1	$10 \leq w \leq 15 \wedge c = 5$	(FF1) or (FF2)
UF2	$10 \leq w \leq 15 \wedge c = 10$	(FF3) or (FF4)
UF3	$10 \leq w \leq 15 \wedge c = 15$	(FF5) or (FF6)
UF4	$10 \leq w \leq 15 \wedge c = 20$	(FF7) or (FF8)
UF5	$25 < w < 40 \wedge c = 15$	(FF9)
UF6	$25 < w \leq 40 \wedge c = 20$	(FF11)
UF7	$25 < w < 40 \wedge c = 20$	(FF12)
UF8	$25 < w \leq 30 \wedge c = 15$	(FF13)
UF9	$25 < w < 30 \wedge c = 15$	(FF15)
UF10	$25 < w \leq 30 \wedge c = 20$	(FF17)
UF11	$25 < w < 30 \wedge c = 20$	(FF18)
UF12	$15 < w < 40 \wedge c = 20$	(FF24) or (FF25)
UF13	$15 < w \leq 30 \wedge c = 20$	(FF26) or (FF25)
UF14	$15 < w < 30 \wedge c = 20$	(FF27) or (FF25)
UF15	$10 < w < 40 \wedge c = 20$	(FF24) or (FF28) or (FF29)
UF16	$10 < w < 40 \wedge c = 20$	(FF24) or (FF28) or (FF30)

Table B.3: Symbolic states computed for the location Urgent off.

Code	Constraint	Targets
UN1	$5 \leq w < 10 \wedge c = 5$	(NF7)
UN2	$0 \leq w < 10 \wedge c = 10$	(NF8)
UN3	$0 \leq w < 10 \wedge c = 15$	(NF9)
UN4	$0 \leq w < 10 \wedge c = 20$	(NF10)
UN5	$5 \leq w < 25 \wedge c = 10$	(NF11) or (NF12) or (NF13)
UN6	$5 \leq w \leq 25 \wedge c = 15$	(NF14) or (NF15) or (NF16)
UN7	$5 \leq w \leq 25 \wedge c = 20$	(NF17) or (NF18) or (NF19)
UN8	$0 \leq w < 25 \wedge c = 15$	(NF20) or (NF15) or (NF21)
UN9	$0 \leq w \leq 25 \wedge c = 20$	(NF17) or (NF18) or (NF22)
UN10	$0 \leq w < 25 \wedge c = 20$	(NF23) or (NF18) or (NF24)
UN11	$0 \leq w \leq 25 \wedge c = 15$	(NF14) or (NF15) or (NF21)
UN12	$5 \leq w < 10 \wedge c = 20$	(NF24)

Table B.4: Symbolic states computed for the location Urgent on.

Code	Constraint	Targets
FF1	$15 - 2t \leq w \leq 15 \wedge g_l = 15 - 2t \wedge g_u = 20 \wedge 5 \leq c \leq 10$	(N11,NF1)
FF2	$10 - 2t \leq w < 15 \wedge g_l = 10 - 2t \wedge g_u = 15 \wedge 5 \leq c \leq 10$	(N12,NF2)
FF3	$15 - 2t \leq w \leq 15 \wedge g_l = 15 - 2t \wedge g_u = 20 \wedge 10 \leq c \leq 15$	(N14,NF3)
FF4	$10 - 2t \leq w < 15 \wedge g_l = 10 - 2t \wedge g_u = 15 \wedge 10 \leq c \leq 15$	(N15,NF4)
FF5	$15 - 2t \leq w \leq 15 \wedge g_l = 15 - 2t \wedge g_u = 20 \wedge 15 \leq c \leq 20$	(N16,NF5)
FF6	$10 - 2t \leq w < 15 \wedge g_l = 10 - 2t \wedge g_u = 15 \wedge 15 \leq c \leq 20$	(N17,NF6)
FF7	$15 - 2t \leq w \leq 15 \wedge g_l = 15 - 2t \wedge g_u = 20 \wedge 20 \leq c \leq 25$	TARGET
FF8	$10 - 2t \leq w < 15 \wedge g_l = 10 - 2t \wedge g_u = 15 \wedge 20 \leq c \leq 25$	TARGET
FF9	$25 - 2t < w < 40 \wedge g_l = 25 - 2t \wedge g_u = 40 \wedge 15 \leq c \leq 20$	(F24,FF10)
FF10	$15 - 2t < w < 40 \wedge g_l = 15 - 2t \wedge g_u = 40 \wedge 20 \leq c \leq 25$	TARGET
FF11	$25 - 2t < w \leq 40 \wedge g_l = 25 - 2t \wedge g_u = 40 \wedge 20 \leq c \leq 25$	TARGET
FF12	$25 - 2t < w < 40 \wedge g_l = 25 - 2t \wedge g_u = 40 \wedge 20 \leq c \leq 25$	TARGET
FF13	$25 - 2t < w \leq 30 \wedge g_l = 25 - 2t \wedge g_u = 40 \wedge 15 \leq c \leq 20$	(F11,FF14)
FF14	$15 - 2t < w \leq 30 \wedge g_l = 15 - 2t \wedge g_u = 40 \wedge 20 \leq c \leq 25$	TARGET
FF15	$25 - 2t < w < 30 \wedge g_l = 25 - 2t \wedge g_u = 40 \wedge 15 \leq c \leq 20$	(F13,FF16)
FF16	$15 - 2t < w < 30 \wedge g_l = 15 - 2t \wedge g_u = 40 \wedge 20 \leq c \leq 25$	TARGET
FF17	$25 - 2t < w \leq 30 \wedge g_l = 25 - 2t \wedge g_u = 40 \wedge 20 \leq c \leq 25$	TARGET
FF18	$25 - 2t < w < 30 \wedge g_l = 25 - 2t \wedge g_u = 40 \wedge 20 \leq c \leq 25$	TARGET
FF19	$25 - 2t < w < 40 \wedge g_l = 20 - 2t \wedge g_u = 40 \wedge 15 \leq c \leq 20$	(F7,FF20)
FF20	$10 - 2t < w < 40 \wedge g_l = 10 - 2t \wedge g_u = 40 \wedge 20 \leq c \leq 25$	TARGET
FF21	$15 - 2t \leq w \leq 35 \wedge g_l = 15 - 2t \wedge g_u = 35 \wedge 15 \leq c \leq 20$	SHUTDOWN
FF22	$20 - 2t < w \leq 40 \wedge g_l = 20 - 2t \wedge g_u = 40 \wedge 20 \leq c \leq 25$	TARGET
FF23	$15 - 2t \leq w \leq 35 \wedge g_l = 15 - 2t \wedge g_u = 35 \wedge 20 \leq c \leq 25$	TARGET
FF24	$20 - 2t < w < 40 \wedge g_l = 20 - 2t \wedge g_u = 40 \wedge 20 \leq c \leq 25$	TARGET
FF25	$15 - 2t < w \leq 20 \wedge g_l = 15 - 2t \wedge g_u = 20 \wedge 20 \leq c \leq 25$	TARGET
FF26	$20 - 2t < w \leq 30 \wedge g_l = 20 - 2t \wedge g_u = 40 \wedge 20 \leq c \leq 25$	TARGET
FF27	$20 - 2t < w < 30 \wedge g_l = 20 - 2t \wedge g_u = 40 \wedge 20 \leq c \leq 25$	TARGET
FF28	$15 - 2t \leq w \leq 20 \wedge g_l = 15 - 2t \wedge g_u = 20 \wedge 20 \leq c \leq 25$	TARGET
FF29	$10 - 2t < w < 15 \wedge g_l = 10 - 2t \wedge g_u = 15 \wedge 20 \leq c \leq 25$	TARGET
FF30	$10 - 2t \leq w < 15 \wedge g_l = 20 - 2t \wedge g_u = 15 \wedge 20 \leq c \leq 25$	TARGET

Table B.5: Symbolic states computed for the location Off/failure.

Code	Constraint	Targets
NF1	$0 \leq w \leq 15+3t \wedge g_l = 5 \wedge g_u = 20+3t \wedge 10 \leq c \leq 15$	SHUTDOWN
NF2	$0 \leq w < 15+3t \wedge g_l = 0 \wedge g_u = 15+3t \wedge 10 \leq c \leq 15$	SHUTDOWN
NF3	$0 \leq w \leq 15+3t \wedge g_l = 5 \wedge g_u = 20+3t \wedge 15 \leq c \leq 20$	SHUTDOWN
NF4	$0 \leq w < 15+3t \wedge g_l = 0 \wedge g_u = 15+3t \wedge 15 \leq c \leq 20$	SHUTDOWN
NF5	$0 \leq w \leq 15+3t \wedge g_l = 5 \wedge g_u = 20+3t \wedge 20 \leq c \leq 25$	TARGET
NF6	$0 \leq w < 15+3t \wedge g_l = 0 \wedge g_u = 15+3t \wedge 20 \leq c \leq 25$	TARGET
NF7	$5 \leq w < 10+3t \wedge g_l = 0 \wedge g_u = 15+3t \wedge 5 \leq c \leq 10$	SHUTDOWN
NF8	$0 \leq w < 10+3t \wedge g_l = 0 \wedge g_u = 15+3t \wedge 10 \leq c \leq 15$	SHUTDOWN
NF9	$0 \leq w < 10+3t \wedge g_l = 0 \wedge g_u = 15+3t \wedge 15 \leq c \leq 20$	SHUTDOWN
NF10	$0 \leq w < 10+3t \wedge g_l = 0 \wedge g_u = 15+3t \wedge 20 \leq c \leq 25$	TARGET
NF11	$20 \leq w < 25+3t \wedge g_l = 20 \wedge g_u = 25+3t \wedge 10 \leq c \leq 15$	(F16,FF19)
NF12	$15 \leq w \leq 20+3t \wedge g_l = 15 \wedge g_u = 20+3t \wedge 10 \leq c \leq 15$	(F19,FF21)
NF13	$5 \leq w < 15+3t \wedge g_l = 0 \wedge g_u = 15+3t \wedge 10 \leq c \leq 15$	SHUTDOWN
NF14	$20 < w \leq 25+3t \wedge g_l = 20 \wedge g_u = 25+3t \wedge 15 \leq c \leq 20$	(F21,FF22)
NF15	$15 \leq w \leq 20+3t \wedge g_l = 15 \wedge g_u = 20+3t \wedge 15 \leq c \leq 20$	(F22,FF23)
NF16	$5 \leq w < 15+3t \wedge g_l = 0 \wedge g_u = 15+3t15 \leq c \leq 20$	SHUTDOWN
NF17	$20 < w \leq 25+3t \wedge g_l = 20 \wedge g_u = 25+3t \wedge 20 \leq c \leq 25$	TARGET
NF18	$15 \leq w \leq 20+3t \wedge g_l = 15 \wedge g_u = 20+3t \wedge 20 \leq c \leq 25$	TARGET
NF19	$5 \leq w < 15+3t \wedge g_l = 0 \wedge g_u = 15+3t \wedge 20 \leq c \leq 25$	TARGET
NF20	$20 < w < 25+3t \wedge g_l = 20 \wedge g_u = 25+3t \wedge 15 \leq c \leq 20$	(F23,FF24)
NF21	$0 \leq w < 15+3t \wedge g_l = 0 \wedge g_u = 15+3t \wedge 15 \leq c \leq 20$	SHUTDOWN
NF22	$0 \leq w < 15+3t \wedge g_l = 0 \wedge g_u = 15+3t \wedge 20 \leq c \leq 25$	TARGET
NF23	$20 < w < 25+3t \wedge g_l = 20 \wedge g_u = 25+3t \wedge 20 \leq c \leq 25$	TARGET
NF24	$0 \leq w < 15+3t \wedge g_l = 0 \wedge g_u = 15+3t \wedge 20 \leq c \leq 25$	TARGET
NF25	$5 \leq w < 10+3t \wedge g_l = 0 \wedge g_u = 15+3t \wedge 20 \leq c \leq 25$	TARGET

Table B.6: Symbolic states computed for the location On/failure.

tuple of the chosen distribution. These transitions are obtained from the distributions of the probabilistic rectangular automata R_{sb} in a natural way; as the generation of a set of symbolic states occurs when the edges of a particular distribution are considered, we take this set of symbolic states as the support of a distribution, the probabilities of which are obtained from those of the distribution of R_{sb}.

Note that, as the variables g_l and g_u are constant in the locations Off, On, Urgent off and Urgent on, we omit all reference to their values in the constraints describing the polyhedra in these locations. Also note that the value of the clock t is always 0 in the locations Urgent off and Urgent on, and is always subject to the constraint $0 \leq t \leq \Delta$ in Off or On. Therefore, we do not make reference to the clock t in the following *except* in the cases in which a constraint on the water level or either bound of the estimate is dependent on t.

B.2 PRISM: input and output files

B.2.1 Introduction

Although this is not the place for an in-depth explanation as to the workings of the probabilistic model checking tool PRISM [dAKN$^+$00], we highlight a number of important points. Recall that the input language of PRISM is that of *probabilistic modules*, which are a probabilistic-nondeterministic variant of the reactive modules of Alur and Henzinger [AH99]. In our setting the concurrent probabilistic system $\mathcal{S}_{R_{sb}}^{Target}$ comprises only one module. A module can have several discrete variables associated with it, a valuation of which makes up a state. In our setting we have three variables: to understand their use, first recall that we uniquely identified each symbolic state of the forward reachability graph $\mathcal{S}_{R_{sb}}^{Target}$ by a code consisting of letters and a number. Then we let the variable `l` correspond to the letter of the symbolic state, which in turn refers to the symbolic state's location. As we let the variable `l` be an enumerated integer with the range 1,...,6, we associate the value 1 with the location Off and the code F, 2 with the location On and the code N, 3 with Urgent off and UF, 4 with Urgent on and UN, 5 with Off/failure and FF, and, finally, 6 is associated with the location On/failure and the code NF. To represent the number of the code of a symbolic state, we use the variable `n`. The third variable `t` is used to encode whether the symbolic state is dashed (`= 3`), shaded (`= 2`) or normal (`= 1`). Recall that the dashed symbolic states correspond to those that can reach the location Shutdown in one transition with probability 1, whereas the shaded symbolic states correspond to those that can reach the target in one transition. The global system is allowed to possess a number of constants, which we use to represent the probability values of discrete transitions. A state, and its transitions, is coded in the following way:

```
[] (l=2) & (n=3) & (t=1) -> e:(l'=2) & (n'=4) & (t'=2) + g:(l'=4) & (n'=7) & (t'=1);
[] (l=2) & (n=3) & (t=1) -> h:(l'=1) & (n'=8) & (t'=2) + i:(l'=3) & (n'=6) & (t'=1);
```

As the variable `l` equals 2, the above refers to the transition relation of the symbolic state denoted by N, which is a "normal" symbolic state (it does not correspond to a Shutdown or reaching the target location), as `t` is equal to 1. It makes a transition according to a nondeterministic choice between two distributions, as denoted by the two lines of code each starting with `[]`.. The first line includes the information concerning its associated distribution after the `->`, which specifies that, with probability `e`, we make a transition to a state with `l` equal to 2, `n` equal to 4, and `t` equal to 2. Similarly, with probability `g`, we make a transition to a state with `l`, `n` and `t` equal to 4, 7 and 1 respectively. The distribution given by the second line is denoted in a similar way. Observing that comments follow a pair of forward slashes `//`, we refer the reader to Section B.2.2 for the full probabilistic module code of the concurrent probabilistic system representing the forward reachability graph $\mathcal{S}_{R_{sb}}^{Target}$.

As stated in Section 9.5.2, the input format of PRISM requirement properties takes the form of PCTL formulae. Recall that the PCTL property of interest to us is $\neg\mathbb{P}_{<0.98}(\texttt{true}\ \mathcal{U}\ \textit{shaded})$. As the *shaded* states of Figure 9.1 correspond to the value of the variable `t` being 2, this PCTL formula is coded in the requirement property language of PRISM as $!([\texttt{trueU}(\texttt{t} = 2)] < \texttt{0.98})$. The output produced by PRISM given the input of this formula, and the probabilistic module given in Section B.2.2, is presented in Section B.2.3.

B.2.2 Probabilistic module for the steam boiler

Following is the probabilistic module code for the concurrent probabilistic system given in Figure 9.1. In combination with the PCTL formula also given in Section 9.5.2, this code can be used as input for the probabilistic model checking tool Prism.

```
// steam boiler example

nondeterministic

// constants for probabilities

prob a = 0.9; // F to F
prob b = 0.1; // F to UF

prob c = 0.9; // F to N
prob d = 0.1; // F to UN

prob e = 0.9; // N to N
prob g = 0.1; // N to UN

prob h = 0.9; // N to F
prob i = 0.1; // N to UF

prob m = 0.9; // FF to N
prob o = 0.1; // FF to NF

prob k = 0.9; // FF to F
prob j = 0.1; // FF to FF

prob s = 0.9; // NF to F
prob r = 0.1; // NF to FF

// main module

module boiler

l : [1..6];
n : [1..31];
t : [1..3];


// l=1 (Location: Off)

[] (l=1) & (n=1) & (t=1) -> a:(l'=1) & (n'=2) & (t'=1) + b:(l'=3) & (n'=1) & (t'=1);
[] (l=1) & (n=1) & (t=1) -> c:(l'=2) & (n'=1) & (t'=1) + d:(l'=4) & (n'=1) & (t'=1);

[] (l=1) & (n=2) & (t=1) -> a:(l'=1) & (n'=3) & (t'=1) + b:(l'=3) & (n'=2) & (t'=1);
[] (l=1) & (n=2) & (t=1) -> c:(l'=2) & (n'=5) & (t'=1) + d:(l'=4) & (n'=2) & (t'=1);

[] (l=1) & (n=3) & (t=1) -> a:(l'=1) & (n'=4) & (t'=1) + b:(l'=3) & (n'=3) & (t'=1);
[] (l=1) & (n=3) & (t=1) -> c:(l'=2) & (n'=8) & (t'=1) + d:(l'=4) & (n'=3) & (t'=1);

[] (l=1) & (n=4) & (t=1) -> a:(l'=1) & (n'=5) & (t'=2) + b:(l'=3) & (n'=4) & (t'=1);
[] (l=1) & (n=4) & (t=1) -> c:(l'=2) & (n'=10) & (t'=2) + d:(l'=4) & (n'=4) & (t'=1);
[] (l=1) & (n=5) & (t=2) -> 1:(l'=1) & (n'=5) & (t'=2);

[] (l=1) & (n=6) & (t=1) -> a:(l'=1) & (n'=7) & (t'=2) + b:(l'=3) & (n'=12) & (t'=1);

[] (l=1) & (n=7) & (t=2) -> 1:(l'=1) & (n'=7) & (t'=2);

[] (l=1) & (n=8) & (t=2) -> 1:(l'=1) & (n'=8) & (t'=2);

[] (l=1) & (n=9) & (t=2) -> 1:(l'=1) & (n'=9) & (t'=2);

[] (l=1) & (n=10) & (t=1) -> a:(l'=1) & (n'=11) & (t'=2) + b:(l'=3) & (n'=13) & (t'=1);

[] (l=1) & (n=11) & (t=2) -> 1:(l'=1) & (n'=11) & (t'=2);

[] (l=1) & (n=12) & (t=1) -> a:(l'=1) & (n'=13) & (t'=2) + b:(l'=3) & (n'=14) & (t'=1);

[] (l=1) & (n=13) & (t=2) -> 1:(l'=1) & (n'=13) & (t'=2);

[] (l=1) & (n=14) & (t=2) -> 1:(l'=1) & (n'=14) & (t'=2);

[] (l=1) & (n=15) & (t=2) -> 1:(l'=1) & (n'=15) & (t'=2);

[] (l=1) & (n=16) & (t=2) -> 1:(l'=1) & (n'=16) & (t'=2);

[] (l=1) & (n=17) & (t=1) -> a:(l'=1) & (n'=18) & (t'=2) + b:(l'=3) & (n'=15) & (t'=1);

[] (l=1) & (n=18) & (t=2) -> 1:(l'=1) & (n'=18) & (t'=2);

[] (l=1) & (n=19) & (t=1) -> a:(l'=1) & (n'=20) & (t'=2) + b:(l'=3) & (n'=16) & (t'=1);
[] (l=1) & (n=19) & (t=1) -> c:(l'=2) & (n'=18) & (t'=2) + d:(l'=4) & (n'=12) & (t'=1);

[] (l=1) & (n=20) & (t=2) -> 1:(l'=1) & (n'=20) & (t'=2);

[] (l=1) & (n=21) & (t=2) -> 1:(l'=1) & (n'=21) & (t'=2);

[] (l=1) & (n=22) & (t=2) -> 1:(l'=1) & (n'=22) & (t'=2);

[] (l=1) & (n=23) & (t=2) -> 1:(l'=1) & (n'=23) & (t'=2);

// l=2 (Location: On)

[] (l=2) & (n=1) & (t=1) -> e:(l'=2) & (n'=2) & (t'=1) + g:(l'=4) & (n'=5) & (t'=1);

[] (l=2) & (n=2) & (t=1) -> e:(l'=2) & (n'=3) & (t'=1) + g:(l'=4) & (n'=6) & (t'=1);
[] (l=2) & (n=2) & (t=1) -> h:(l'=1) & (n'=6) & (t'=1) + i:(l'=3) & (n'=5) & (t'=1);

[] (l=2) & (n=3) & (t=1) -> e:(l'=2) & (n'=4) & (t'=2) + g:(l'=4) & (n'=7) & (t'=1);
[] (l=2) & (n=3) & (t=1) -> h:(l'=1) & (n'=8) & (t'=2) + i:(l'=3) & (n'=6) & (t'=1);

[] (l=2) & (n=4) & (t=2) -> 1:(l'=2) & (n'=4) & (t'=2);

[] (l=2) & (n=5) & (t=1) -> e:(l'=2) & (n'=6) & (t'=1) + g:(l'=4) & (n'=8) & (t'=1);

[] (l=2) & (n=6) & (t=1) -> e:(l'=2) & (n'=7) & (t'=2) + g:(l'=4) & (n'=9) & (t'=1);
[] (l=2) & (n=6) & (t=1) -> h:(l'=1) & (n'=9) & (t'=2) + i:(l'=3) & (n'=7) & (t'=1);

[] (l=2) & (n=7) & (t=2) -> 1:(l'=2) & (n'=7) & (t'=2);

[] (l=2) & (n=8) & (t=1) -> e:(l'=2) & (n'=9) & (t'=2) + g:(l'=4) & (n'=10) & (t'=1);
[] (l=2) & (n=9) & (t=2) -> 1:(l'=2) & (n'=9) & (t'=2);

[] (l=2) & (n=10) & (t=2) -> 1:(l'=2) & (n'=10) & (t'=2);

[] (l=2) & (n=11) & (t=1) -> e:(l'=2) & (n'=3) & (t'=1) + g:(l'=4) & (n'=6) & (t'=1);
[] (l=2) & (n=11) & (t=1) -> h:(l'=1) & (n'=10) & (t'=1) + i:(l'=3) & (n'=8) & (t'=1);

[] (l=2) & (n=12) & (t=1) -> e:(l'=2) & (n'=13) & (t'=1) + g:(l'=4) & (n'=11) & (t'=1);
[] (l=2) & (n=12) & (t=1) -> h:(l'=1) & (n'=12) & (t'=1) + i:(l'=3) & (n'=9) & (t'=1);

[] (l=2) & (n=13) & (t=1) -> e:(l'=2) & (n'=7) & (t'=2) + g:(l'=4) & (n'=9) & (t'=1);
[] (l=2) & (n=13) & (t=1) -> h:(l'=1) & (n'=14) & (t'=2) + i:(l'=3) & (n'=6) & (t'=1);

[] (l=2) & (n=14) & (t=1) -> e:(l'=2) & (n'=7) & (t'=2) + g:(l'=4) & (n'=9) & (t'=1);
[] (l=2) & (n=14) & (t=1) -> h:(l'=1) & (n'=15) & (t'=2) + i:(l'=3) & (n'=10) & (t'=1);

[] (l=2) & (n=15) & (t=1) -> e:(l'=2) & (n'=7) & (t'=2) + g:(l'=4) & (n'=9) & (t'=1);
[] (l=2) & (n=15) & (t=1) -> h:(l'=1) & (n'=16) & (t'=2) + i:(l'=3) & (n'=11) & (t'=1);

[] (l=2) & (n=16) & (t=2) -> 1:(l'=2) & (n'=16) & (t'=2);

[] (l=2) & (n=17) & (t=2) -> 1:(l'=2) & (n'=17) & (t'=2);

[] (l=2) & (n=18) & (t=2) -> 1:(l'=2) & (n'=18) & (t'=2);

// l=3 (Location: Urgent off)

[] (l=3) & (n=1) & (t=1) -> 1:(l'=5) & (n'=1) & (t'=1);
[] (l=3) & (n=1) & (t=1) -> 1:(l'=5) & (n'=2) & (t'=1);

[] (l=3) & (n=2) & (t=1) -> 1:(l'=5) & (n'=3) & (t'=1);
[] (l=3) & (n=2) & (t=1) -> 1:(l'=5) & (n'=4) & (t'=1);

[] (l=3) & (n=3) & (t=1) -> 1:(l'=5) & (n'=5) & (t'=1);
[] (l=3) & (n=3) & (t=1) -> 1:(l'=5) & (n'=6) & (t'=1);

[] (l=3) & (n=4) & (t=1) -> 1:(l'=5) & (n'=7) & (t'=2);
[] (l=3) & (n=4) & (t=1) -> 1:(l'=5) & (n'=8) & (t'=2);

[] (l=3) & (n=5) & (t=1) -> 1:(l'=5) & (n'=9) & (t'=1);

[] (l=3) & (n=6) & (t=1) -> 1:(l'=5) & (n'=11) & (t'=2);

[] (l=3) & (n=7) & (t=1) -> 1:(l'=5) & (n'=12) & (t'=2);

[] (l=3) & (n=8) & (t=1) -> 1:(l'=5) & (n'=13) & (t'=1);

[] (l=3) & (n=9) & (t=1) -> 1:(l'=5) & (n'=15) & (t'=1);

[] (l=3) & (n=10) & (t=1) -> 1:(l'=5) & (n'=17) & (t'=2);

[] (l=3) & (n=11) & (t=1) -> 1:(l'=5) & (n'=18) & (t'=2);

[] (l=3) & (n=12) & (t=1) -> 1:(l'=5) & (n'=25) & (t'=2);
[] (l=3) & (n=12) & (t=1) -> 1:(l'=5) & (n'=26) & (t'=2);

[] (l=3) & (n=13) & (t=1) -> 1:(l'=5) & (n'=26) & (t'=2);
[] (l=3) & (n=13) & (t=1) -> 1:(l'=5) & (n'=27) & (t'=2);

[] (l=3) & (n=14) & (t=1) -> 1:(l'=5) & (n'=26) & (t'=2);
[] (l=3) & (n=14) & (t=1) -> 1:(l'=5) & (n'=28) & (t'=2);
[] (l=3) & (n=15) & (t=1) -> 1:(l'=5) & (n'=29) & (t'=2);
[] (l=3) & (n=15) & (t=1) -> 1:(l'=5) & (n'=30) & (t'=2);

[] (l=3) & (n=16) & (t=1) -> 1:(l'=5) & (n'=25) & (t'=2);
[] (l=3) & (n=16) & (t=1) -> 1:(l'=5) & (n'=29) & (t'=2);
[] (l=3) & (n=16) & (t=1) -> 1:(l'=5) & (n'=31) & (t'=2);

// l=4 (Location: Urgent on)

[] (l=4) & (n=1) & (t=1) -> 1:(l'=6) & (n'=7) & (t'=3);

[] (l=4) & (n=2) & (t=1) -> 1:(l'=6) & (n'=8) & (t'=3);

[] (l=4) & (n=3) & (t=1) -> 1:(l'=6) & (n'=9) & (t'=3);

[] (l=4) & (n=4) & (t=1) -> 1:(l'=6) & (n'=10) & (t'=2);

[] (l=4) & (n=5) & (t=1) -> 1:(l'=6) & (n'=11) & (t'=1);
[] (l=4) & (n=5) & (t=1) -> 1:(l'=6) & (n'=12) & (t'=1);
[] (l=4) & (n=5) & (t=1) -> 1:(l'=6) & (n'=13) & (t'=3);

[] (l=4) & (n=6) & (t=1) -> 1:(l'=6) & (n'=14) & (t'=1);
[] (l=4) & (n=6) & (t=1) -> 1:(l'=6) & (n'=15) & (t'=1);
[] (l=4) & (n=6) & (t=1) -> 1:(l'=6) & (n'=16) & (t'=3);

[] (l=4) & (n=7) & (t=1) -> 1:(l'=6) & (n'=17) & (t'=2);
[] (l=4) & (n=7) & (t=1) -> 1:(l'=6) & (n'=18) & (t'=2);
[] (l=4) & (n=7) & (t=1) -> 1:(l'=6) & (n'=19) & (t'=2);

[] (l=4) & (n=8) & (t=1) -> 1:(l'=6) & (n'=20) & (t'=1);
[] (l=4) & (n=8) & (t=1) -> 1:(l'=6) & (n'=15) & (t'=1);
[] (l=4) & (n=8) & (t=1) -> 1:(l'=6) & (n'=21) & (t'=3);

[] (l=4) & (n=9) & (t=1) -> 1:(l'=6) & (n'=17) & (t'=2);
[] (l=4) & (n=9) & (t=1) -> 1:(l'=6) & (n'=18) & (t'=2);
[] (l=4) & (n=9) & (t=1) -> 1:(l'=6) & (n'=22) & (t'=2);

[] (l=4) & (n=10) & (t=1) -> 1:(l'=6) & (n'=23) & (t'=2);
[] (l=4) & (n=10) & (t=1) -> 1:(l'=6) & (n'=18) & (t'=2);
[] (l=4) & (n=10) & (t=1) -> 1:(l'=6) & (n'=24) & (t'=2);

[] (l=4) & (n=11) & (t=1) -> 1:(l'=6) & (n'=14) & (t'=1);
[] (l=4) & (n=11) & (t=1) -> 1:(l'=6) & (n'=15) & (t'=1);
[] (l=4) & (n=11) & (t=1) -> 1:(l'=6) & (n'=21) & (t'=3);

[] (l=4) & (n=12) & (t=1) -> 1:(l'=6) & (n'=25) & (t'=2);

// l=5 (Location: Off/failure)

[] (l=5) & (n=1) & (t=1) -> m:(l'=2) & (n'=11) & (t'=1) + o:(l'=6) & (n'=1) & (t'=3);

[] (l=5) & (n=2) & (t=1) -> m:(l'=2) & (n'=12) & (t'=1) + o:(l'=6) & (n'=2) & (t'=3);

[] (l=5) & (n=3) & (t=1) -> m:(l'=2) & (n'=14) & (t'=1) + o:(l'=6) & (n'=3) & (t'=3);

[] (l=5) & (n=4) & (t=1) -> m:(l'=2) & (n'=15) & (t'=1) + o:(l'=6) & (n'=4) & (t'=3);

[] (l=5) & (n=5) & (t=1) -> m:(l'=2) & (n'=16) & (t'=2) + o:(l'=6) & (n'=5) & (t'=2);

[] (l=5) & (n=6) & (t=1) -> m:(l'=2) & (n'=17) & (t'=2) + o:(l'=6) & (n'=6) & (t'=2);
[] (l=5) & (n=7) & (t=2) -> 1:(l'=5) & (n'=7) & (t'=2);

[] (l=5) & (n=8) & (t=2) -> 1:(l'=5) & (n'=8) & (t'=2);

[] (l=5) & (n=9) & (t=1) -> k:(l'=1) & (n'=23) & (t'=2) + j:(l'=5) & (n'=10) & (t'=2);

[] (l=5) & (n=10) & (t=2) -> 1:(l'=5) & (n'=10) & (t'=2);

[] (l=5) & (n=11) & (t=2) -> 1:(l'=5) & (n'=11) & (t'=2);

[] (l=5) & (n=12) & (t=2) -> 1:(l'=5) & (n'=12) & (t'=2);

[] (l=5) & (n=13) & (t=1) -> k:(l'=1) & (n'=11) & (t'=2) + j:(l'=5) & (n'=14) & (t'=2);

[] (l=5) & (n=14) & (t=2) -> 1:(l'=5) & (n'=14) & (t'=2);

[] (l=5) & (n=15) & (t=1) -> k:(l'=1) & (n'=13) & (t'=2) + j:(l'=5) & (n'=16) & (t'=2);

[] (l=5) & (n=16) & (t=2) -> 1:(l'=5) & (n'=16) & (t'=2);

[] (l=5) & (n=17) & (t=2) -> 1:(l'=5) & (n'=17) & (t'=2);

[] (l=5) & (n=18) & (t=2) -> 1:(l'=5) & (n'=18) & (t'=2);

[] (l=5) & (n=19) & (t=1) -> k:(l'=1) & (n'=7) & (t'=2) + j:(l'=5) & (n'=20) & (t'=2);

[] (l=5) & (n=20) & (t=2) -> 1:(l'=5) & (n'=20) & (t'=2);

[] (l=5) & (n=21) & (t=2) -> 1:(l'=5) & (n'=21) & (t'=2);

[] (l=5) & (n=22) & (t=2) -> 1:(l'=5) & (n'=22) & (t'=2);

[] (l=5) & (n=23) & (t=2) -> 1:(l'=5) & (n'=23) & (t'=2);

[] (l=5) & (n=24) & (t=2) -> 1:(l'=5) & (n'=24) & (t'=2);

[] (l=5) & (n=25) & (t=2) -> 1:(l'=5) & (n'=25) & (t'=2);

[] (l=5) & (n=26) & (t=2) -> 1:(l'=5) & (n'=26) & (t'=2);

[] (l=5) & (n=27) & (t=2) -> 1:(l'=5) & (n'=27) & (t'=2);

[] (l=5) & (n=28) & (t=2) -> 1:(l'=5) & (n'=28) & (t'=2);

[] (l=5) & (n=29) & (t=2) -> 1:(l'=5) & (n'=29) & (t'=2);

[] (l=5) & (n=30) & (t=2) -> 1:(l'=5) & (n'=30) & (t'=2);

[] (l=5) & (n=31) & (t=2) -> 1:(l'=5) & (n'=31) & (t'=2);

// l=6 (Location: On/failure)

[] (l=6) & (n=1) & (t=3) -> 1:(l'=6) & (n'=1) & (t'=3);

[] (l=6) & (n=2) & (t=3) -> 1:(l'=6) & (n'=2) & (t'=3);

[] (l=6) & (n=3) & (t=3) -> 1:(l'=6) & (n'=3) & (t'=3);

[] (l=6) & (n=4) & (t=3) -> 1:(l'=6) & (n'=4) & (t'=3);
[] (l=6) & (n=5) & (t=2) -> 1:(l'=6) & (n'=5) & (t'=2);

[] (l=6) & (n=6) & (t=2) -> 1:(l'=6) & (n'=6) & (t'=2);

[] (l=6) & (n=7) & (t=3) -> 1:(l'=6) & (n'=7) & (t'=3);

[] (l=6) & (n=8) & (t=3) -> 1:(l'=6) & (n'=8) & (t'=3);

[] (l=6) & (n=9) & (t=3) -> 1:(l'=6) & (n'=9) & (t'=3);

[] (l=6) & (n=10) & (t=2) -> 1:(l'=6) & (n'=10) & (t'=2);

[] (l=6) & (n=11) & (t=1) -> s:(l'=1) & (n'=17) & (t'=1) + r:(l'=5) & (n'=19) & (t'=1);

[] (l=6) & (n=12) & (t=1) -> s:(l'=1) & (n'=19) & (t'=1) + r:(l'=5) & (n'=21) & (t'=2);

[] (l=6) & (n=13) & (t=3) -> 1:(l'=6) & (n'=13) & (t'=3);

[] (l=6) & (n=14) & (t=1) -> s:(l'=1) & (n'=21) & (t'=2) + r:(l'=5) & (n'=22) & (t'=2);

[] (l=6) & (n=15) & (t=1) -> s:(l'=1) & (n'=22) & (t'=2) + r:(l'=5) & (n'=23) & (t'=2);

[] (l=6) & (n=16) & (t=3) -> 1:(l'=6) & (n'=16) & (t'=3);

[] (l=6) & (n=17) & (t=2) -> 1:(l'=6) & (n'=17) & (t'=2);

[] (l=6) & (n=18) & (t=2) -> 1:(l'=6) & (n'=18) & (t'=2);

[] (l=6) & (n=19) & (t=2) -> 1:(l'=6) & (n'=19) & (t'=2);

[] (l=6) & (n=20) & (t=1) -> s:(l'=1) & (n'=22) & (t'=2) + r:(l'=5) & (n'=24) & (t'=2);

[] (l=6) & (n=21) & (t=3) -> 1:(l'=6) & (n'=21) & (t'=3);

[] (l=6) & (n=22) & (t=2) -> 1:(l'=6) & (n'=22) & (t'=2);

[] (l=6) & (n=23) & (t=2) -> 1:(l'=6) & (n'=23) & (t'=2);

[] (l=6) & (n=24) & (t=2) -> 1:(l'=6) & (n'=24) & (t'=2);

[] (l=6) & (n=25) & (t=2) -> 1:(l'=6) & (n'=25) & (t'=2);

endmodule
```

B.2.3 Output of Prism

We now present an abbreviated form of the output of the probabilistic model checking tool Prism, given that inputs of the probabilistic module of the previous section and the PCTL formula given in Section 9.5.2.

Note that the list of values after the word `Probabilities` refers to the states and the maximum probabilities with which they satisfy the until formula (`true` $\mathcal{U}$ *shaded*), provided that the probability is nonzero. States are encoded as 3-vectors, in which the first element refers to the value of the variable `l`, the second to `n`, and the final of which refers to `t`.

```
Prism
=====

Version: 1.1
Engine:  MTBDD

Initialising...

Parsing modules file "boiler.nm"...

Building model...

Computing reachable states...

Reachability took 7 iterations and 0.01 seconds.

Model built in 0.576 seconds.

Type:       Nondeterministic
Modules:    boiler
Variables: l n t

States: 125
Transition matrix: 672 nodes (4 terminal), 640 minterms

Parsing file "boiler.pctl"...

PCTL Formula: !([ 1 U t=2 ] < 0.98)

    PCTL Until:

    b1 = 125 states, b2 = 52 states

    Prob1E algorithm took 22 iterations and 0.03 seconds
    Prob0E algorithm took 3 iterations and 0.0 seconds

    yes = 104 states, no = 13 states, maybe = 8 states

    Method took 5 iterations and 0.01 seconds
    (set up 0.0, average 0.0020 per iteration)
    Probabilities (1):
(1,1,1):0.981  (1,2,1):0.99  (1,3,1):1.0  (1,4,1):1.0  (1,5,2):1.0
(1,6,1):1.0    (1,7,2):1.0   (1,8,2):1.0  (1,9,2):1.0  (1,10,1):1.0
(1,11,2):1.0   (1,12,1):1.0  (1,13,2):1.0 (1,14,2):1.0 (1,15,2):1.0
(1,16,2):1.0   (1,17,1):1.0  (1,18,2):1.0 (1,19,1):1.0 (1,20,2):1.0
(1,21,2):1.0   (1,22,2):1.0  (1,23,2):1.0 (2,1,1):1.0  (2,2,1):1.0
(2,3,1):1.0    (2,4,2):1.0   (2,5,1):1.0  (2,6,1):1.0  (2,7,2):1.0
(2,8,1):1.0    (2,9,2):1.0   (2,10,2):1.0 (2,11,1):1.0 (2,12,1):1.0
(2,13,1):1.0   (2,14,1):1.0  (2,15,1):1.0 (2,16,2):1.0 (2,17,2):1.0
(2,18,2):1.0   (3,1,1):0.9   (3,2,1):0.9  (3,3,1):1.0  (3,4,1):1.0
(3,5,1):1.0    (3,6,1):1.0   (3,7,1):1.0  (3,8,1):1.0  (3,9,1):1.0
(3,10,1):1.0   (3,11,1):1.0  (3,12,1):1.0 (3,13,1):1.0 (3,14,1):1.0
(3,15,1):1.0   (3,16,1):1.0  (4,4,1):1.0  (4,5,1):1.0  (4,6,1):1.0
(4,7,1):1.0    (4,8,1):1.0   (4,9,1):1.0  (4,10,1):1.0 (4,11,1):1.0
(4,12,1):1.0   (5,1,1):0.9   (5,2,1):0.9  (5,3,1):0.9  (5,4,1):0.9
(5,5,1):1.0    (5,6,1):1.0   (5,7,2):1.0  (5,8,2):1.0  (5,9,1):1.0
(5,10,2):1.0   (5,11,2):1.0  (5,12,2):1.0 (5,13,1):1.0 (5,14,2):1.0
(5,15,1):1.0   (5,16,2):1.0  (5,17,2):1.0 (5,18,2):1.0 (5,19,1):1.0
(5,20,2):1.0   (5,21,2):1.0  (5,22,2):1.0 (5,23,2):1.0 (5,24,2):1.0
(5,25,2):1.0   (5,26,2):1.0  (5,27,2):1.0 (5,28,2):1.0 (5,29,2):1.0
(5,30,2):1.0   (5,31,2):1.0  (6,5,2):1.0  (6,6,2):1.0  (6,10,2):1.0
(6,11,1):1.0   (6,12,1):1.0  (6,14,1):1.0 (6,15,1):1.0 (6,17,2):1.0
(6,18,2):1.0   (6,19,2):1.0  (6,20,1):1.0 (6,22,2):1.0 (6,23,2):1.0
(6,24,2):1.0   (6,25,2):1.0

    Number of states satisfying until: 918

Model checking completed in 0.114 seconds.

Number of states satisfying PCTL formula: 106
```

Therefore, the initial symbolic state F1 is associated with the probability 0.981, hence verifying our manual calculation obtained in Section 9.5.2. The reader can also verify that the other manually calculated probabilities, as labelling the appropriate states of Figure 9.2, tally with the results obtained by using Prism.

We note that Prism also outputs the list of states that satisfy the overall formula, but we omit this here, as the satisfaction of the formula $\neg\mathbb{P}_{<0.98}(\texttt{true}\,\mathcal{U}\,\textit{shaded})$ can be derived easily from the list of probabilities given in the output above.

Bibliography

[ABL96] J.-R. Abrial, E. Börger, and H. Langmaack, editors. *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165 of *Lecture Notes in Computer Science State-of-the-Art Survey*. Springer-Verlag, 1996.

[ABL98] L. Aceto, A. Burgueño, and K. G. Larsen. Model checking via reachability testing for timed automata. In B. Steffen, editor, *Proceedings of the 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, volume 1384 of *Lecture Notes in Computer Science*, pages 263–280. Springer-Verlag, 1998.

[Abr96] J.-R. Abrial. Steam-boiler control specification problem, 1996. In Abrial, Börger and Langmaack [ABL96].

[ACD91a] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking for probabilistic real-time systems. In J. Leach Albert, B. Monien, and M. Artalejo Rodríguez, editors, *Proceedings of the 18th International Conference on Automata, Languages and Programming (ICALP'91)*, volume 510 of *Lecture Notes in Computer Science*, pages 115–136. Springer-Verlag, 1991.

[ACD91b] R. Alur, C. Courcoubetis, and D. L. Dill. Verifying automata specifications of probabilistic real-time systems. In J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors, *Proceedings of Real-Time: Theory in Practice, REX Workshop*, volume 600 of *Lecture Notes in Computer Science*, pages 28–44. Springer-Verlag, 1991.

[ACD+92a] R. Alur, C. Courcoubetis, D. L. Dill, N. Halbwachs, and H. Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. In R. Werner, editor, *Proceedings of the 13th IEEE Real-Time Systems Symposium (RTSS'92)*, pages 157–166. IEEE Computer Society Press, 1992.

[ACD+92b] R. Alur, C. Courcoubetis, D. L. Dill, N. Halbwachs, and H. Wong-Toi. Minimization of timed transition systems. In W. R. Cleaveland, editor, *Proceedings of the 3rd Conference on Concurrency Theory (CONCUR '92)*, volume 630 of *Lecture Notes in Computer Science*, pages 340–354. Springer-Verlag, 1992.

[ACD93] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[ACH$^+$95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[ACH97] R. Alur, C. Courcoubetis, and T. A. Henzinger. Computing accumulated delays in real-time systems. *Formal Methods in System Design*, 11(2):137–156, 1997.

[ACHH93] R. Alur, C. Courcoubetis, T. A. Henzinger, and P. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In R. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Proceedings of the 1st International Workshop on Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer-Verlag, 1993.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AD96] R. Alur and D. L. Dill. Automata-theoretic verification of real-time systems. In C. Heitmeyer and D. Mandrioli, editors, *Formal Methods for Real-Time Computing*, Trends in Software, pages 58–82. John Wiley & Sons Ltd, 1996.

[AEK$^+$99] R. Alur, J. Esposito, M. Kim, V. Kumar, and I. Lee. Formal modeling and analysis of hybrid systems: A case study in multi-robot coordination. In J. M. Wing, J. Woodcock, and J. Davies, editors, *Proceedings of the World Congress on Formal Methods (FM'99), Volume I*, volume 1708 of *Lecture Notes in Computer Science*, pages 212–232. Springer-Verlag, 1999.

[AH99] R. Alur and T. A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.

[AHH96] R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.

[AHLP00] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, July 2000. Special issue on Hybrid Systems: Theory & Applications.

[AHV93] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *Proceedings of the 25th ACM Symposium on Theory of Computing (STOC'93)*, pages 592–601. ACM Press, 1993.

[ASB$^+$95] A. Aziz, V. Singhal, F. Balarin, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In P. Wolper, editor, *Proceedings of the 7th International Conference On Computer Aided Verification (CAV'95)*, volume 939 of *Lecture Notes in Computer Science*, pages 155–165. Springer-Verlag, 1995.

[ASSB96] A. Aziz, K. Sanwal, V. Singhal, and R. K. Brayton. Verifying continuous-time Markov chains. In R. Alur and T. A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 269–276. Springer-Verlag, 1996.

[Bai96] C. Baier. Polynomial-time algorithms for testing probabilistic bisimulation and simulation. In Rajeev Alur and T. A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 50–61. Springer-Verlag, 1996.

[Bai98] C. Baier. On algorithmic verification methods for probabilistic systems, 1998. Habilitation thesis, University of Mannheim.

[BBC$^+$96] N. Bjorner, A. Browne, E. Chang, M. Colon, A. Kapur, Z. Manna, H. B. Sipma, and T. E. Uribe. STeP: deductive-algorithmic verification of reactive and real-time systems. In R. Alur and T. A. Henzinger, editors, *Proceedings of the 8th International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 415–418. Springer Verlag, 1996.

[BBLS92] S. Bensalem, A. Bouajjani, C. Loiseaux, and J. Sifakis. Property Preserving Simulations. In G. V. Bochmann and D. K. Probst, editors, *Proceedings of the 4th Workshop on Computer-Aided Verification (CAV'92)*, volume 663 of *Lecture Notes in Computer Science*, pages 260–273, 1992.

[BCDM86] M. C. Browne, E. M. Clarke, D. L. Dill, and B. Mishra. Automatic verification of sequential circuits using temporal logic. *IEEE Transactions on Computers*, C-35(12):1035–1043, 1986.

[BCG88] M. C. Browne, E. M. Clarke, and O. Grümberg. Characterizing finite Kripke structures in Propositional Temporal Logic. *Theoretical Computer Science*, 59(1-2):115–131, July 1988.

[BCH$^+$97] C. Baier, E. M. Clarke, V. Hartonas-Garmhausen, M. Z. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proceedings of the 24th International Conference on Automata, Languages and Programming (ICALP'97)*, volume 1256 of *Lecture Notes in Computer Science*, pages 430–440. Springer-Verlag, 1997.

[BCM$^+$92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[BD00] B. Bérard and C. Dufourd. Timed automata and additive clock constraints. *Information Processing Letters*, 75(1–2):1–7, 2000.

[BdA95] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. S. Thiagarajan, editor, *Proceedings of the International Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS'95)*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer-Verlag, 1995.

[BDFP00a] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Are timed automata updatable? In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification (CAV2000)*, volume 1855 of *Lecture Notes in Computer Science*, pages 464–479. Springer-Verlag, 2000.

[BDFP00b] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Expressiveness of updatable timed automata. In M. Nielsen and B. Rovan, editors, *Proceedings of the 25th International Symposium on the Mathematical Foundations of Computer Science (MFCS2000)*, volume 1893 of *Lecture Notes in Computer Science*, pages 232–242. Springer-Verlag, 2000.

[BDM$^+$98] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: a model-checking tool for real-time systems. In A. Hu and M. Vardi, editors, *Proceedings of the 10th Conference on Computer Aided Verification (CAV'98)*, volume 1486, pages 298–302. Springer-Verlag, 1998.

[BEM00] C. Baier, B. Engelen, and M. Majster-Cederbaum. Deciding bisimilarity and similarity for probabilistic processes. *Journal of Computer and System Sciences*, 60(1):187–231, 2000.

[BES93] A. Bouajjani, R. Echahed, and J. Sifakis. On model checking for real-time properties with durations. In *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science (LICS'93)*, pages 147–159. IEEE Computer Society Press, 1993.

[BFH$^+$01] G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. Vaandrager. Minimum-cost reachability for priced timed automata. In M. Domenica Di Benedetto and A. L. Sangiovanni-Vincentelli, editors, *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC2001)*, Lecture Notes in Computer Science. Springer-Verlag, 2001.

[BGK$^+$96] J. Bengtsson, W. O. D. Griffioen, K. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Verification of an audio protocol with bus collision using UPPAAL. In R. Alur and T. A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 244–256. Springer-Verlag, 1996.

[BHHK00] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification (CAV2000)*, volume 1855 of *Lecture Notes in Computer Science*, pages 358–372. Springer-Verlag, 2000.

[BHV00] G. Behrmann, T. Hune, and F. Vaandrager. Distributed timed model checking - How the search order matters. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification (CAV2000)*, volume 1855 of *Lecture Notes in Computer Science*, pages 216–231. Springer-Verlag, 2000.

[Bil86] P. Billingsley. *Probability and Measure*. Probability and Mathematical Statistics. Wiley, 2nd edition, 1986.

[BK98] C. Baier and M. Z. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998.

[BKH99] C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In J. Baeten and S. Mauw, editors, *Proceedings of the 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *Lecture Notes in Computer Science*, pages 142–162. Springer-Verlag, 1999.

[BLY96] A. Bouajjani, Y. Lakhnech, and S. Yovine. Model-checking for extended timed temporal logics. In B. Jonsson and J. Parrow, editors, *Proceedings of the 4th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'96)*, volume 1135, pages 306–326, 1996.

[BMP83] M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.

[BMT99] M. Bozga, O. Maler, and S. Tripakis. Efficient verification of timed automata using dense and discrete time semantics. In L. Pierre and T. Kropf, editors, *Proceedings of 10th Conference on Correct Hardware Design and Verification Methods (CHARME'99)*, volume 1703 of *Lecture Notes in Computer Science*, pages 125–141. Springer-Verlag, 1999.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[BS00] B. Bérard and L. Sierra. Comparing verification with HyTech, Kronos and Uppaal on the railroad crossing example. Technical Report LSV-00-2, Laboratoire Spécification et Vérification, ENS de Cachan, 2000.

[BST00] G. Bandini, R. L. Spelberg, and W. J. Toetenel. Parametric verification of the IEEE1394a root contention protocol using LPMC. Submitted, July 2000.

[BSTV99] A. Burns, R. L. Spelberg, W. J. Toetenel, and T. Vink. Modeling and verification using XTG and PMC. In *Proceedings of the 5th Annual Conference of the Advanced School for Computing and Imaging*, 1999.

[BTY97] A. Bouajjani, S. Tripakis, and S. Yovine. On-the-fly symbolic model checking for real-time systems. In K.-J. Lin and S. H. Son, editors, *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS'97)*. IEEE Computer Society Press, 1997.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In D. Kozen, editor, *Proceedings of the Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CGL94] E. M. Clarke, O. Grümberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

[CGP99] E. M. Clarke, O. Grümberg, and D. Peled. *Model checking.* MIT Press, 1999.

[CK96] E. M. Clarke and R. Kurshan. Computer-aided verification. *IEEE Spectrum*, 33(6):61–67, 1996.

[CL00] F. Cassez and K. G. Larsen. The impressive power of stopwatches. In C. Palamidessi, editor, *Proceedings of 11th International Conference on Concurrency Theory (CONCUR2000)*, volume 1877 of *Lecture Notes in Computer Science*, pages 138–152. Springer-Verlag, 2000.

[CT00] C. Courcoubetis and S. Tripakis. Probabilistic model checking: formalisms and algorithms for discrete and real-time systems. In K. Inan, editor, *Verification of Digital and Hybrid Systems.* NATO-ASI publication, 2000.

[CY88] C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *Proceeding of the 29th Annual Symposium on Foundations of Computer Science (FOCS'88)*, pages 338–345. IEEE Computer Society Press, 1988.

[CY95] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, July 1995.

[CY98] C. Courcoubetis and M. Yannakakis. Markov decision processes and regular events. *IEEE Trans. on Automatic Control*, 43(10):1399–1418, 1998.

[dA97a] L. de Alfaro. *Formal verification of probabilistic systems.* PhD thesis, Stanford University, Department of Computer Science, 1997.

[dA97b] L. de Alfaro. Temporal logics for the specification of performance and reliability. In R. Reishuck and M. Morvan, editors, *Procedings of the 14th Symposium on Theoretical Aspects of Computer Science (STACS'97)*, volume 1200 of *Lecture Notes in Computer Science*, pages 165–176. Springer-Verlag, 1997.

[dA98a] L. de Alfaro. How to specify and verify the long-run average behavior of probabilistic systems. In V. Pratt, editor, *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science (LICS'98)*, pages 454–465. IEEE Computer Society Press, 1998.

[dA98b] L. de Alfaro. Stochastic transition systems. In D. Sangiorgi and R. de Simone, editors, *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 423–438. Springer-Verlag, 1998.

[D'A99a] P. R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. PhD thesis, Department of Computer Science, University of Twente, November 1999.

[dA99b] L. de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In J. Baeten and S. Mauw, editors, *Proceedings of the International Conference on Concurrency Theory 1999 (CONCUR'99)*, volume 1664 of *Lecture Notes in Computer Science*, pages 66–81. Springer-Verlag, 1999.

[D'A00a] P. R. D'Argenio. A compositional translation of stochastic automata into timed automata. Technical Report CTIT 00-08, University of Twente, May 2000.

[D'A00b] P. R. D'Argenio. Personal communication, July 2000.

[dAKN$^+$00] L. de Alfaro, M. Z. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of concurrent probabilistic processes using MTBDDs and the Kronecker representation. In S. Graf and M. Schwartzbach, editors, *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS2000)*, volume 1785 of *Lecture Notes in Computer Science*, pages 395–410. Springer-Verlag, 2000.

[Daw98] C. Daws. Optikron: a tool suite for enhancing model-checking of real-time systems. In A. Hu and M. Y. Vardi, editors, *Proceedings of the 10th Conference on Computer Aided Verification (CAV'98)*, volume 1427, pages 542–545. Springer-Verlag, 1998.

[DC86] D. L. Dill and E. M. Clarke. Automatic verification of asynchronous circuits using temporal logic. *IEE Proceedings*, 133(5):276–282, September 1986.

[DGG97] D. Dams, R. Gerth, and O. Grümberg. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems*, 19(2):253–291, March 1997.

[DGJP00] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Approximating labeled markov processes. In M. Abadi and J. Mitchell, editors, *Proceedings of the 15th Annual Symposium on Logic in Computer Science (LICS'00)*, pages 95–106. IEEE Computer Society Press, 2000.

[Dil89] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer-Verlag, 1989.

[DKB98] P. R. D'Argenio, J.-P. Katoen, and E. Brinksma. An algebraic approach to the specification of stochastic systems (extended abstract). In D. Gries and W.-P. de Roever, editors, *Proceedings of the IFIP Working conference on Programming Concepts and Methods (PROCOMET'98)*, IFIP Series, pages 126–147. Chapman & Hall, 1998.

[DKRT97] P. R. D'Argenio, J.-P. Katoen, T. C. Ruys, and J. Tretmans. The bounded retransmission protocol must be on time! In E. Brinksma, editor, *Proceedings of the Third Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'97)*, volume 1217 of *Lecture Notes in Computer Science*, pages 416–431. Springer-Verlag, 1997.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In R. Alur, T. A. Henzinger, and E. D. Sontag, editors, *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, pages 208–219. Springer-Verlag, 1996.

[DT98] C. Daws and S. Tripakis. Model–checking of real–time reachability properties using abstractions. In B. Steffen, editor, *Proceedings of the 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, volume 1384 of *Lecture Notes in Computer Science*, pages 313–329. Springer-Verlag, 1998.

[DY95] C. Daws and S. Yovine. Two examples of verification of multirate timed automata with KRONOS. In A. Burns, Y.-H. Lee, and K. Ramamritham, editors, *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS'95)*, pages 66–75. IEEE Computer Society Press, 1995.

[DY96] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In A. Burns, Y.-H. Lee, and S.H. Son, editors, *Proceedings of the 1996 IEEE Real-Time Systems Symposium (RTSS'96)*, pages 73–81. IEEE Computer Society Press, 1996.

[EH86] E. A. Emerson and J. Y. Halpern. "Sometimes" and "Not Never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33:151–178, 1986.

[Eme90] E.A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 996–1072, Amsterdam, 1990. Elsevier Science Publishers.

[FGK$^+$96] J.-C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu. CADP: a protocol validation and verification toolbox. In R. Alur and T. A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 437–440. Springer-Verlag, 1996.

[GJ95] H. Gregersen and H. E. Jensen. Formal design of reliable real time systems. Master's thesis, Institute of Electronic Systems, Department of Mathematics and Computer Science, Aalborg University, 1995.

[GL94] O. Grümberg and D. E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, 1994.

[Han94] H. Hansson. *Time and probability in formal design of distributed systems.* Elsevier, 1994.

[Hen95] T. A. Henzinger. Hybrid automata with finite bisimulations. In Z. Fülöp and F. Gécseg, editors, *Proceedings of the 22nd International Conference on Automata, Languages, and Programming (ICALP'95)*, volume 944 of *Lecture Notes in Computer Science*, pages 324–335. Springer-Verlag, 1995.

[Hen96] T. A. Henzinger. The theory of hybrid automata. In E. M. Clarke, J.G. Riecke, and M. Y. Vardi, editors, *Proceedings of the 11th Annual Symposium on Logic in Computer Science (LICS'96)*, pages 278–292. IEEE Computer Society Press, 1996.

[HHK95] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *Proceedings of the 36rd Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 453–462. IEEE Computer Society Press, 1995.

[HHM99] T. A. Henzinger, B. Horowitz, and R. Majumdar. Rectangular hybrid games. In J. C. M. Baeten and S. Mauw, editors, *Proceedings of the 10th International Conference on Concurrency Theory (CONCUR'99)*, volume

1664 of *Lecture Notes in Computer Science*, pages 320–335. Springer-Verlag, 1999.

[HHMW00] T. A. Henzinger, B. Horowitz, R. Majumdar, and H. Wong-Toi. Beyond HyTech: hybrid systems analysis using interval numerical methods. In B. Krogh and N. A. Lynch, editors, *Proceedings of the 3rd International Workshop on Hybrid Systems: Computation and Control (HSCC2000)*, volume 1790 of *Lecture Notes in Computer Science*, pages 130–144. Springer-Verlag, 2000.

[HHW95] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to HyTech. In E. Brinksma, W. R. Cleaveland, K. G. Larsen, T. Margaria, and B. Steffen, editors, *Proceedings of the 1st International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95)*, volume 1019 of *Lecture Notes in Computer Science*, pages 41–71. Springer-Verlag, 1995.

[HHW97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: a model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.

[HHWT98] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):540–554, 1998.

[HJ94] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[HK96] T. A. Henzinger and P. W. Kopke. State equivalences for rectangular hybrid automata. In U. Montanari and V. Sassone, editors, *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96)*, volume 1119 of *Lecture Notes in Computer Science*, pages 530–545. Springer-Verlag, 1996.

[HKMS00] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A Markov chain model checker. In S. Graf and M. Schwartzbach, editors, *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS2000)*, volume 1785 of *Lecture Notes in Computer Science*, pages 347–362. Springer-Verlag, 2000.

[HKPV98] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, August 1998.

[HKV96] T. A. Henzinger, O. Kupferman, and M.Y. Vardi. A space-efficient on-the-fly algorithm for real-time model checking. In U. Montanari and V. Sassone, editors, *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96)*, volume 1119 of *Lecture Notes in Computer Science*, pages 514–529. Springer-Verlag, 1996.

[HLPS99] J. Hu, J. Lygeros, M. Prandini, and S. Sastry. Aircraft conflict detection and resolution using brownian motion. In *Proceedings of the 38th IEEE Conference on Decision and Control*, 1999.

[HLS99] K. Havelund, K. G. Larsen, and A. Skou. Formal verification of a power controller using the real-time model checker UPPAAL. In J.-P. Katoen, editor, *Proceedings of the 5th International AMAST Workshop on Real-Time and Probabilistic Systems (ARTS'99)*, volume 1601 of *Lecture Notes in Computer Science*, pages 277–298. Springer-Verlag, 1999.

[HLS00] J. Hu, J. Lygeros, and S. Sastry. Towards a theory of stochastic hybrid systems. In B. Krogh and N. Lynch, editors, *Proceedings of the 3rd International Workshop on Hybrid Systems: Computation and Control (HSCC2000)*, volume 1790 of *Lecture Notes in Computer Science*, pages 160–173. Springer-Verlag, 2000.

[HM00a] T. A. Henzinger and R. Majumdar. A classification of symbolic transition systems. In H. Reichel and S. Tison, editors, *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS2000)*, volume 1770 of *Lecture Notes in Computer Science*, pages 13–34. Springer-Verlag, 2000.

[HM00b] T. A. Henzinger and R. Majumdar. Symbolic model checking for rectangular hybrid systems. In S. Graf and M. Schwartzbach, editors, *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS2000)*, volume 1785 of *Lecture Notes in Computer Science*, pages 142–156. Springer-Verlag, 2000.

[HMP92] T. A. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Proceedings of Real-Time: Theory in Practice, REX Workshop*, volume 600 of *Lecture Notes in Computer Science*, pages 226–251. Springer-Verlag, 1992.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[Ho95] P.-H. Ho. *Automatic Analysis of Hybrid Systems.* PhD thesis, Cornell University, 1995.

[Hol97] G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[HR98] T. A. Henzinger and V. Rusu. Reachability verification for hybrid automata. In T. A. Henzinger and S. Sastry, editors, *Proceedings of the 1st International Workshop on Hybrid Systems: Computation and Control (HSCC'98)*, volume 1386 of *Lecture Notes in Computer Science*, pages 190–204. Springer-Verlag, 1998.

[HSLL97] K. Havelund, A. Skou, K. G. Larsen, and K. Lund. Formal modelling and analysis of an audio/video protocol: An industrial case study using Uppaal. In K.-J. Lin and S. H. Son, editors, *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS'97)*, pages 2–13. IEEE Computer Society Press, 1997.

[HW95] P.-H. Ho and H. Wong-Toi. Automated analysis of an audio control protocol. In P. Wolper, editor, *Proceedings of the 7th International Conference on Computer-aided Verification (CAV'95)*, number 939 in Lecture Notes in Computer Science, pages 381–394. Springer-Verlag, 1995.

[HW96] T. A. Henzinger and H. Wong-Toi. Using HyTech to synthesize control parameters for a steam boiler. In J.-R. Abrial, E. Börger, and H. Langmaack, editors, *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165 of *Lecture Notes in Computer Science*, pages 265–282. Springer-Verlag, 1996.

[HY00] J. Hou and S. Yovine. An integrated GUI for Kronos and Open Kronos. VHS (Verification of Hybrid Systems) Deliverable TL.1 - Tools, 2000.

[IN96] P. Iyer and M. Narasimha. "Almost always" and "sometime definitely" are not enough: Probabilistic quantifiers and probabilistic model-checking. Technical Report TR-96-16, Department of Computer Science, North Carolina State University, 1996.

[JL91] B. Jonsson and K. G. Larsen. Specification and refinement of probabilistic processes. In A. R. Meyer, editor, *Proceedings of the 6th Annual IEEE Symposium on Logic in Computer Science (LICS'91)*, pages 266–279. IEEE Computer Society Press, 1991.

[KNS00] M. Z. Kwiatkowska, G. Norman, and J. Sproston. Symbolic model checking of probabilistic timed automata using backwards reachability. Technical Report CSR-00-01, University of Birmingham, January 2000.

[KNSS99] M. Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. In J.-P. Katoen, editor, *Proceedings of the 5th International AMAST Workshop on Real-Time and Probabilistic Systems (ARTS'99)*, volume 1601 of *Lecture Notes in Computer Science*, pages 75–95. Springer-Verlag, 1999.

[KNSS00a] M. Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 2000. Special issue on ARTS'99. To appear.

[KNSS00b] M.Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Verifying quantitative properties of continuous probabilistic timed automata. In C. Palamidessi, editor, *Proceedings of the 11th International Conference On Concurrency Theory (CONCUR2000)*, volume 1877 of *Lecture Notes in Computer Science*, pages 123–137. Springer-Verlag, 2000.

[KP95] K. J. Kristoffersen and P. Pettersson. Modelling and analysis of a steam generator using UPPAAL. In B. Bjerner, M. Larsson, and B. Nordström, editors, *Proceedings of the 7th Nordic Workshop on Programming Theory*, pages 156–164, 1995. Report 86, Chalmers University of Technology.

[KPSY99] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Decidable integration graphs. *Information and Computation*, 150(2):209–243, 1999.

[Kri99] K.J. Kristoffersen. *Compositional Verification of Concurrent Systems - A Possible Cure for the State/Explosion Problem.* PhD thesis, Basic Research in Computer Science, Aalborg University, 1999.

[KS83] P. C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1983.

[KVW00] O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

[LP00] K. G. Larsen and P. Pettersson. Lecture notes of 'real-time systems', 2000. Department of Computer Science, Aalborg University.

[LPWY99] K. G. Larsen, J. Pearson, C. Weise, and W. Yi. Clock difference diagrams. *Nordic Journal of Computing*, 6(3):271–298, 1999.

[LPY95a] K. G. Larsen, P. Pettersson, and W. Yi. Compositional and symbolic model-checking of real-time systems. In A. Burns, Y.-H. Lee, and K. Ramamritham, editors, *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS'95)*, pages 76–87. IEEE Computer Society Press, 1995.

[LPY95b] K. G. Larsen, P. Pettersson, and W. Yi. Model checking for real-time systems. In H. Reichel, editor, *Proceedings of the 10th International Conference on Fundamentals of Computation Theory*, number 965 in Lecture Notes in Computer Science, pages 62–88. Springer-Verlag, 1995.

[LPY97a] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Software Tools for Technology Transfer*, 1(1+2):134–152, 1997.

[LPY97b] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL: Status and developments. In O. Grümberg, editor, *Proceedings of the 9th International Conference on Computer Aided Verification (CAV'97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 456–459. Springer-Verlag, 1997.

[LPY98] M. Lindahl, P. Pettersson, and W. Yi. Formal design and analysis of a gear-box controller. In B. Steffen, editor, *Proceedings of the 4th Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, number 1384 in Lecture Notes in Computer Science, pages 281–297. Springer-Verlag, 1998.

[LPY99] G. Lafferriere, G. J. Pappas, and S. Yovine. A new class of decidable hybrid systems. In F. Vaandrager and J. van Schuppen, editors, *Proceedings of the 2nd International Workshop on Hybrid Systems: Computation and Control (HSCC'99)*, volume 1569 of *Lecture Notes in Computer Science*, pages 137–151. Springer-Verlag, 1999.

[LS91] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, September 1991.

[LV96] N. A. Lynch and F. Vaandrager. Forward and backward simulations, II: Timing-based systems. *Information and Computation*, 128(1):1–25, 1996.

[McM93] K. McMillan. *Symbolic model checking: An approach to the state explosion problem.* Kluwer Academic Publishers, 1993.

[Mil89] R. Milner. *Communication and Concurrency.* International Series in Computer Science. Prentice Hall, 1989.

[MMP92] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors, *Proceedings of Real-Time: Theory in Practice, REX Workshop*, volume 600 of *Lecture Notes in Computer Science*, pages 447–484. Springer-Verlag, 1992.

[MMT98] A. McIver, C. Morgan, and E. Troubitsyna. The probabilistic steam boiler: a case study in probabilistic data refinement. In J. Grundy, M. Schwenke, and T. Vickers, editors, *Proceedings of the International Refinement Workshop and Formal Methods Pacific (IRW/FMP'98)*, Discrete Mathematics and Theoretical Computer Science. Springer-Verlag, 1998.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.

[MY96] O. Maler and S. Yovine. Hardware timing verification using KRONOS. In *Proceedings of the 7th IEEE Israeli Conference on Computer Systems and Software Engineering (ICCBSSE'96)*. IEEE Computer Society Press, 1996.

[NOSY93] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An approach to the description and analysis of hybrid systems. In R. Grossman, A. Nerode, A. Ravn, and H. Rischel, editors, *Proceedings of the 1st International Workshop on Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 149–178. Springer-Verlag, 1993.

[NS91] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebra. In K. G. Larsen and A. Skou, editors, *Procedings of the 3rd International Conference on Computer Aided Verification (CAV'91)*, volume 575 of *Lecture Notes in Computer Science*, pages 376–398. Springer-Verlag, 1991.

[NS95] S. Nadjm-Tehrani and J.-E. Strömberg. Modelling and formal analysis of an aircraft landing gear system. In A. Bouajjani and O. Maler, editors, *Proceedings of the 2nd. European workshop on Real-time and Hybrid Systems*, pages 239–246, 1995.

[NTY00] P. Niebert, S. Tripakis, and S. Yovine. Minimum-time reachability for timed automata. In *Proceedings of the 8th IEEE Mediteranean Control Conference*. IEEE, 2000. To appear.

[OSY94] A. Olivero, J. Sifakis, and S. Yovine. Using abstractions for the verification of linear hybrid systems. In D. L. Dill, editor, *Proceedings of the*

6th International Conference on Computer-Aided Verification (CAV'94), volume 818 of *Lecture Notes in Computer Science*, pages 81–94. Springer-Verlag, 1994.

[PL00] P. Pettersson and K. G. Larsen. Uppaal2k. *Bulletin of the European Association for Theoretical Computer Science*, 70:40–44, 2000.

[PLNS99] M. Prandini, J. Lygeros, A. Nilim, and S. Sastry. A probabilistic framework for aircraft conflict detection. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, 1999.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, 1977.

[PT87] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[PV94] A. Puri and P. Varaiya. Decidability of hybrid systems with rectangular differential inclusions. In D.L. Dill, editor, *Proceedings of the 6th International Conference on Computer-Aided Verification (CAV'94)*, volume 818 of *Lecture Notes in Computer Science*, pages 95–104. Springer-Verlag, 1994.

[PZ86] A. Pnueli and L. D. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1:53–72, 1986.

[PZ93] A. Pnueli and L. D. Zuck. Probabilistic verification. *Information and Computation*, 103(1):1–29, 1993.

[QS82] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In M. Dezani-Ciancaglini and U. Montanari, editors, *Proceedings of the International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, 1982.

[RR96] O. Roux and V. Rusu. Uniformity for the decidability of hybrid automata. In R. Cousot and D. A. Schmidt, editors, *Proceedings of the International Static Analysis Symposium (SAS'96)*, volume 1145 of *Lecture Notes in Computer Science*, pages 301–316. Springer-Verlag, 1996.

[Seg95] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, MIT, Dept. of Electrical Engineering and Computer Science, 1995.

[SL95] R. Segala and N. A. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[SMF97] T. Stauner, O. Müller, and M. Fuchs. Using HyTech to verify an automotive control system. In O. Maler, editor, *Proceedings of the International Workshop on Hybrid and Real-Time Systems (HART'97)*, volume 1201 of *Lecture Notes in Computer Science*, pages 139–153. Springer-Verlag, 1997.

[Spr99] J. Sproston. Analyzing subclasses of probabilistic hybrid automata. In M. Z. Kwiatkowska, editor, *Proceedings of the 2nd International Workshop on Probabilistic Methods in Verification (PROBMIV'99)*, pages 67–91, 1999. Proceedings appeared as Technical Report CSR-99-8, School of Computer Science, The University of Birmingham.

[Spr00] J. Sproston. Decidable model checking of probabilistic hybrid automata. In M. Joseph, editor, *Proceedings of the 6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT2000)*, volume 1926 of *Lecture Notes in Computer Science*, pages 31–45. Springer-Verlag, 2000.

[SS95] O. V. Sokolsky and S. A. Smolka. Local model checking for real-time systems. In P. Wolper, editor, *Proceedings of the 7th International Conference on Computer Aided Verification (CAV'95)*, volume 939 of *Lecture Notes in Computer Science*, pages 211–224. Springer-Verlag, 1995.

[SS00] D. P. L. Simons and M. I. A. Stoelinga. Mechanical verification of the IEEE1394a root contention protocol using Uppaal2k. Report CSI-R009, Computing Science Institute, University of Nijmegen, Nijmegen, 2000. Submitted.

[STA98] R. Lutje Spelberg, H. Toetenel, and M. Ammerlaan. Partition refinement in real-time model checking. In A.P. Ravn and H. Rischel, editors, *Proceedings of the 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98)*, volume 1486 of *Lecture Notes in Computer Science*, pages 143–157. Springer-Verlag, 1998.

[SV99] M. I. A. Stoelinga and F.W. Vaandrager. Root contention in IEEE1394. In Joost-Pieter Katoen, editor, *ARTS'99: Proceedings of the 5th International AMAST Workshop on Real-Time and Probabilistic Systems*, volume 1601 of *Lecture Notes in Computer Science*, pages 53–74. Springer Verlag, 1999.

[Tri98] S. Tripakis. *L'Analyse Formelle des Systèmes Temporisés en Pratique*. PhD thesis, Université Joseph Fourier, 1998.

[Tri99a] S. Tripakis. Timed diagnostics for reachability properties. In W. R. Cleaveland, editor, *Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *Lecture Notes in Computer Science*, pages 59–73. Springer-Verlag, 1999.

[Tri99b] S. Tripakis. Verifying progress in timed systems. In J.-P. Katoen, editor, *Proceedings of the 5th International AMAST Workshop on Real-Time and Probabilistic Systems (ARTS'99)*, volume 1601 of *Lecture Notes in Computer Science*, pages 299–314. Springer-Verlag, 1999.

[TY98] S. Tripakis and S. Yovine. Verification of the Fast Reservation Protocol with Delayed Transmission using the tool KRONOS. In R. Rajkumar and A. Bestavros, editors, *Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium (RTAS'98)*, pages 165–170. IEEE Computer Society Press, 1998.

[TY00] S. Tripakis and S. Yovine. Analysis of timed systems based on time-abstracting bisimulations. *Formal Methods in System Design*, 2000. To appear.

[Var85] M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science (FOCS'85)*. IEEE Computer Society Press, 1985.

[Var96] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: structure versus automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266, 1996.

[vGSS95] R. J. van Glabbeek, S. A. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 1995.

[VW86] M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Science*, 32(2):183–221, 1986.

[VWB$^+$98] T. Villa, H. Wong-Toi, A. Balluchi, J. Preußig, A. L. Sangiovanni-Vincentelli, and Y. Watanabe. Formal verification of an automotive engine controller in cutoff mode. In J. D. Birdwell and D. A. Castanon, editors, *Proceedings of 37th IEEE Conference on Decision and Control (CDC'98)*. IEEE Computer Society Press, 1998.

[Yi90] W. Yi. Real time behaviour of asynchronous agents. In J. Baeten and J. W. Klop, editors, *Proceedings of the 1st International Conference on Concurrency Theory (CONCUR'90)*, volume 458 of *Lecture Notes in Computer Science*, pages 502–520. Springer-Verlag, 1990.

[YL93] M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition systems. In C. Courcoubetis, editor, *Proceedings of the 5th International Conference on Computer Aided Verification (CAV'93)*, volume 697 of *Lecture Notes in Computer Science*, pages 210–224. Springer-Verlag, 1993.

[Yov97] S. Yovine. Kronos: A verification tool for real-time systems. *Springer International Journal of Software Tools for Technology Transfer*, 1(1/2), 1997.

[Yov98] S. Yovine. Model checking timed automata. In G. Rozenberg and F. Vaandrager, editors, *Embedded Systems*, volume 1494 of *Lecture Notes in Computer Science*, pages 114–152. Springer-Verlag, 1998.

[Yov00] S. Yovine. A brief summary of the tool Kronos. VHS (Verification of Hybrid Systems) Deliverable TL.1 - Tools, 2000.