Compositionality & Abstraction in verification of Probabilistic Transition Systems

Antoinette Ahiable and Tu Hoang Anh
Kim G. Larsen(Supervisor)

Aalborg University
Faculty of Engineering & Science
Department of Computer Science

Compositionality & Abstraction in verification of Probabilistic Transition Systems

Semester Period:
Spring 2003
February 3, 2003 - June 11, 2003

Project Group:
SSE4, B1-215

Authors:
Antoinette Ahiable, atnets@cs.auc.dk
Tu Hoang Anh, tuhoang@cs.auc.dk

Supervisor:
Kim G. Larsen, kgl@cs.auc.dk

Number Of Pages: 103

Total Number of Copies: 6

Abstract

In this thesis, we present techniques adapted to probabilistic transition systems in order to avoid or reduce the state space explored in verification. By using compositional abstraction, abstracts of components of a concurrent system can be used for model checking. We use probabilistic simulation preorder to establish a good abstract. We further present a minimization algorithm for probabilistic transition systems, which generates a minimized structure, with respect to simulation equivalence. Finally, we introduce the Quotient technique for our PTS model with the concept of using the minimization algorithm to minimize the size of the transformed specifications. However, it is realized that this situation never arises, as the transformed specification is the minimal structure possible given our constraints and assumptions. We propose at the end, the application of the simplification heuristic on components of the system, before applying the Quotienting technique rather than afterwards. We implement the quotient technique as part of our tool, CAPS and demonstrate the algorithms proposed in this thesis.

Preface

This report is the thesis in our final semester of the International Masters Program in Software Systems Engineering of the Faculty of Engineering & Sciences, in the Computer Science Department at Aalborg University, Denmark, during the period February 3, 2003 to June 11, 2003, under the Distributed Systems and Semantics group.

This Master thesis is a joint project by the authors and is the result of work done over the last year of our program.

We will like to sincerely thank our supervisor Kim G. Larsen for everything! It says it all.

Antoinette Ahiable		Tu Hoang Anh

________________		________________

Contents

Chapter 1

Introduction

1.1 Formal Verification

Formal verification methods are strong tools in the development of high quality products. In the design of complex systems, more time and effort is spent on verification than on construction. Techniques are sought to reduce and ease the verification efforts while increasing their coverage.
Formal methods offer a large potential to obtain an early integration of verification in the design process, to provide more effective verification techniques and to reduce the verification time. As according to J.-P Katoen in ([33]), formal methods are one of the "highly recommended" verification techniques for software development of safety-critical systems according to e.g., the best practises standards by the IEA (International Electrotechnical Commission) and the standards by the ESA (European Space Agency).

Formal methods provide a precise notion between systems and their specifications, so that it can be decided without ambiguity whether or not a system meets its specification. They, however have their advantages and disadvantages. Comparatively, model checking is automatic and faster than theorem provers. A major problem, though, in applying model checking even to moderate-size systems is the potential combinatorial explosion of the state space arising from parallel composition of components.

When I use a model checker, it runs and runs forever and never comes back ... when I use a static analysis tool it comes back immediately and says I don't know - Patrick Cousot

1.1.1 Model Checking

One of the acclaimed approaches of verifying finite state systems is that of model checking. It is a verification technique that explores all possible system states in a brute force manner. Once a system can be accurately represented by a model, a specification can be verified within this model and conclusions drawn about the system as to whether the specification holds to be true or otherwise within the system. This technique has been applied to many types of systems from finite state representations, through real time ones to probabilistic systems. However, the main disadvantage of the model checking technique is the explosion in the state space during its brute force exploration.

1.1.2 State Explosion

The size of the a parallel system of even moderate-sized systems grows exponentially. In order to avoid this inherent problem of model checkers, several methods have been sought that avoid the exhaustive state space exploration.

Some of these methods are based on a symbolic representation of the system using Binary Decision Diagrams ([14], [40]), which has proved to be very successful for various types of verification problems for parallel systems. Other methods are based on the concept of partial order reduction ([43], [24]) which is based on the observation that the interleaved execution of independent actions allows one to investigate only a representative fragment of the state space.

The concept of compositionality, where the motivation is to reason about the behavior of a large system based on knowledge of its components has also influenced some of the methods. In those cases where a global investigation can be avoided efficiency is gained. In ([42]), compositional reasoning has proven to be a successful technique in the verification of concurrent systems and embedded software, ([10]). Another very significant alternative is by abstractions. This method seeks to use abstractions of the model under consideration, which are smaller than the originals in model checking, hopefully decreasing the time involved and minimizing memory used. Model abstraction reduces the number of states necessary to perform formal verification and thus reduces the state space to be explored in formal verification tools such as COSPAN.

Another approach, which was first proposed by Larsen ([38]) and further extended by Andersen([10]), Bodentien et al ([9]), introduced a very promising heuristic model checking technique for finite state systems called the Quotient Technique. The idea behind it, is to factor out components of a parallel system, one at a time, into the specification, and by continuously, applying simplification heuristics, minimize the resulting structure. By transforming the specification accordingly, one is able to draw conclusions about the model and the specification. This technique has also been applied in real time systems ([35]). In this thesis, we explore this technique with finite state systems and extend it to our probabilistic model. Our aim is to establish the basis for using the Quotient Technique in probabilistic transition systems and also to experimentally verify if there are indeed optimal results using this technique.

1.2 Probabilistic Models

Due to the fact that a system can not always be guaranteed to work correctly, there is the need for a way of describing the unreliability of a system. This is especially important in safety critical systems such as flight control systems and medical systems. This has led to more models being considered with probabilities incorporated. Probabilistic models are important for the quantitative design and analysis of safety critical systems. They are also useful for the analysis of quantitative behavior in a wide variety of systems e.g. through the computation of performance measures.

1.2.1 Markovian Models

The idea of incorporating probabilities into the modelled system has lead to various developments. In recent years, many researchers have focused on reasoning about probabilistic transition systems. A lot of work has been done to extend those models and methods which have been successful for the non-probabilistic case to probabilistic systems. The basic concept of all these models however, is that of a transition system which has been extended appropriately with probabilities. They can be classified though, with respect to their treatment of non-determinism. This has been according to either Markov chains where non-determinism is completely replaced by probabilistic choices or Markov decision processes, in which both non-determinism and probabilities are present. The models based on Markov chains are suitable to formalize the behavior of sequential randomized algorithms or processes of probabilistic calculi with synchronous parallel composition. On the other hand Markov decision processes based models are suitable for distributed randomized algorithms or processes of asynchronous probabilistic calculus. In our work, we choose to use Markov Decision Processes also known as Probabilistic Transition Systems, simply PTS, as our main model.

There are different variants of probabilistic systems and we present these in Section 1.2.2.

1.2.2 Probabilistic Model Classifications

We present work in the field of probabilistic transition systems. Probabilistic transition systems provide a framework that allows us to express that a failure can only occur with a certain probability, and as a tool it can be used to verify that a system, with some probability, behaves according to its specification.(i.e. there is a 0.0002% possibility that a medical monitor will shut down without a warning). There are three main classifications: Reactive, Generative and Stratified models.

Reactive Model

This model consists of states and labelled transitions associated with probabilities. For each state, the sum of probabilities on outgoing transitions must be 1 for transitions with the same label.

Larsen and Skou ([39]) define a reactive probabilistic transition system as follows:

Definition 1 *A reactive probabilistic transition system is a structure $P = (Pr, Act, \pi)$, where Pr is a set of processes(or states), Act is the set of actions that the process may perform, and π is a transition probability function $\pi : Pr \times Act \times Pr \to [0, 1]$ such that for each $P \in Pr$ and $a \in Act$:*

$$\sum_{P' \in Pr} \pi(P, a, P') = 1 \quad \textit{or} \sum_{P' \in Pr} \pi(P, a, P') = 0$$

indicating the possible next states and their probabilities after P has performed action a.

In Figure 1.1 is an example of a reactive process.

Generative Model

This model consists also of states and labelled transitions with probabilities, but with the sum of probabilities of all outgoing transitions equal to 1. Jou and Smolka ([32]) have formally defined the Generative model as follows:

Definition 2 *A generative probabilistic transition system is a triple $\langle Pr, \Sigma, \mu\rangle$, where Pr is a set of processes; Σ is the set of all atomic actions and 0 is a special symbol not in Σ called the zero action; $\mu : (Pr \times (\Sigma \cup 0 \times Pr) \to [0, 1]$ is a total function called the probabilistic transition function satisfying the following restriction: $\forall P \in Pr$,*

$$\sum_{a \in \Sigma \cup 0, Q \in Pr} \mu(P, a, Q) = 1$$

Figure 1.2 is an example of a generative process.

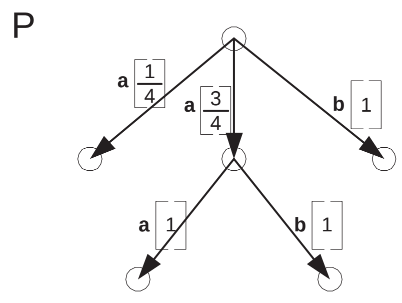


Figure 1.1: An example of a reactive process

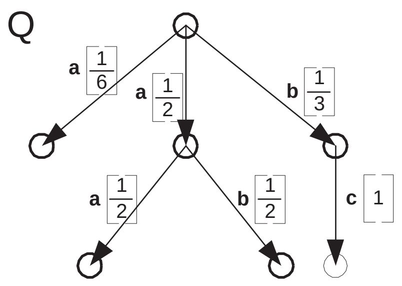


Figure 1.2: An example of a generative process

Stratified Model

Stratified models consists of states and two kinds of transitions, probabilistic and action based. In the case of probabilistic transitions, the sum of probabilities must be 1, and for the action transitions the restriction is that there must be only one outgoing action transition from a state. van Glabbeek et al ([23]) have a well documented paper on these models. They show that the generative model is an abstraction of the stratified model, and that the reactive model is an abstraction of the generative model.

1.3 Specifications

The specification or property to be verified in a model, is usually stated in some logic as a formula. This in the non-probabilistic case could be a Linear Tree Logic (LTL, [18]) or Computational Tree Logic (CTL, [19]) or some temporal logic. More suited to your model of a probabilistic model, is the Probabilistic Computational Tree Logic (PCTL, [12], [25]) which can express quantitative bounds on the probability of system evolutions. We give the syntax and semantics of this logic in Section 2.3. However, there is an alternative to this logic representation of formulas of the specification in model checking.

There are two main approaches for specifying properties in model checking: logic-based (eg CTL, [19]) and automaton-based(eg $\omega - automata$, [34]) or transition system based. In the latter case, specification formalism satisfiability will normally be given by some suitable behavioral equivalence or preorder between the implementation and specification.

We are interested in a restricted class of reachability properties, and hence interested in particular sequences of execution that lead to certain final conditions. These properties allow to specify that the probability of reaching a particular final condition ϕ_f from any reachable state satisfying a given initial condition ϕ_i is smaller (or greater) than a given probability p.

1.4 Topics of Thesis

1.4.1 Related Work

In this section, we first mention some of the existing work in area of methods and techniques to avoid the state space exploration during model checking.

Minimization

The main idea, in Minimization, is to reduce the size of the model used in model checking by algorithms, so that the reduced(and smaller) structure is used in place of the original. By this the state space to be explored, is reduced before model checking. However, the minimization is carried out with respect to the specification in question and by preserving the relations that hold on the models. Orna Grumberg's ([16]) minimization technique is applied to Kripke structures by upholding the simulation preorder.

Compositional Abstractions

In our previous work, ([1]), we described a procedure in which, a system made up of components interacting together, will have each component being replaced by a good abstraction, based on the probabilistic simulation preorder. These component-based abstractions then replace the originals in the model checking procedure. The implementation is a tool, CAPS, which, checks if two probabilistic transition systems (a model and its abstract) simulate each other, and hence the abstract is used in place of the original.

The Quotient Technique

A new approach towards compositional verification of concurrent systems is the Quotient Technique where components are gradually removed from the concurrent system while transforming the specification accordingly. The intermediate specification is kept small using heuristics for minimization. This technique has been used for state/event systems, ([10]) and also , some version of probabilistic systems ([46]).

1.4.2 Outline of thesis

This thesis is organized as such: The next chapter we introduce some basic definitions and theorems in Probability Theory, focus on the Probabilistic Transition Systems, which is our main working model, describe the formalism for our specifications, and state what equivalences and preorders we use in this work.
Chapter 3 is dedicated to Networks and Maximum Flows and how this is used in computing the probabilistic simulation preorder. This chapter is basically work down in our last report.
In Chapter 4, we discuss some general methods for compositionality and abstractions. In Chapter 5, we discuss the Minimization Method and The Quotient Technique for finite states and in Chapter 6 and 7, we focus on these two methods, and propose algorithms adapted to our probabilistic model, respectively.
Chapter 8 discusses the application of these methods in the verification process.
Chapter 9, describes the implementation of these methods in our existing tool, CAPS. Some tests are used in Chapter 10 to draw some experimental results, and Chapter 11, draws conclusions on the work done in this thesis with some future directions.

Chapter 2

Preliminaries

2.1 Probability Theory

In this section we give some basic definitions associated with the term probability that we find useful in this work.

A sample space S lists all the possible outcomes of a random phenomenon. In general the event (E) is a subset of a sample space, or in other words, an event is any collection of outcomes.

The probability function π is a function from the sample space S to a number between 0 and 1.

$$\pi : S \to [0, 1]$$

For each event A in an experiment of a sample space S, $\pi(A)$ is the probability that A will occur. The probability value, $\pi(A)$, assigned to an outcome (event) must satisfy the three axioms below in order to satisfy the mathematical notion of probability.

Axiom 1 : *For any event A, $\pi(A) \geq 0$.*

Axiom 2 : $\pi(S) = 1$.

Axiom 3 : *For an infinite sequence of disjoint events $A_1, A_2, \ldots$*

$$\pi\Big(\bigcup_{i=1}^{\infty} A_i\Big) = \sum_{i=1}^{\infty} \pi(A_i).$$

Definition 3 *A probability distribution or probability on a sample space S is a specification of the numbers $\pi(A)$ which satisfies Axioms 1,2, and 3.*

Consequently, for any event A, $0 \leq \pi(A) \leq 1$.

Generally, if the occurrence of an event A does not influence the occurrence of another event B, it is said that the two events A and B are independent. If events A and B are independent, then $\pi(\text{A and B}) = \pi(A) \cdot \pi(B)$.

2.2 Probabilistic Transition Systems

In this thesis, systems are described in terms of Markov Decision Processes ([29]), also called Probabilistic Transition Systems (PTS). This model is a labelled transition system with both non-determinism and probabilistic choices present. The choice of this model is partly due to the fact that it is closed under parallel composition (which facilitates modelling and compositional reasoning) but primarily because PTSs are amenable to abstraction. This is a key factor for the techniques proposed in this work.

We give a formal definition as such:

Definition 4 *A Probabilistic Transition System PTS is a tuple (S, $\longrightarrow$, V) where*

- *S is a non-empty finite set of states*
- *$\rightarrow \subseteq S \times Act \times Dist(S)$, is a finite transition relation where Act is a finite set of actions, Dist(S) is a distribution over states S,*
- *$V : S \rightarrow 2^{AP}$ is a labelling function.*

Note should be taken of the action-labelled transitions. In the cases where an action is implicit and the same through out a system we leave it out of the representation.

We use $s \xrightarrow{a} \pi$ to denote $(s, a, \pi) \in\longrightarrow$ and $s \overset{a}{\not\longrightarrow}$ to denote that $(s, a, \pi) \notin\longrightarrow$ for all π. A Rooted PTS is a PTS with a predefined initial state, $(S, \longrightarrow, V, s_0)$. Figure 2.1 is an example of a PTS with an initial state s.

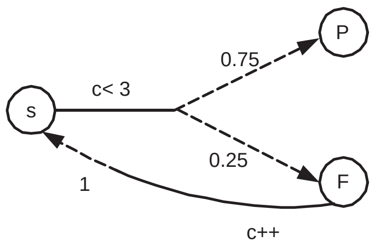


Figure 2.1: A Probabilistic Transition System

2.2.1 Probabilistic Executions

Each action α leads to a distribution $\pi \in Dist(S)$ over successor states. We refer to $s \xrightarrow{\alpha} \pi$ as a transition. Finite processes have finite number of states and the transition relation is acyclic. A PTS is a *Fully Probabilistic Transition System* (FPTS) if whenever $s \xrightarrow{\alpha} \pi$ and $s \xrightarrow{\alpha} \rho$ then $\pi = \rho$.

Let $T = (S, \rightarrow, V)$. A simple path starting from $s_0 \in S$ in T is a finite sequence of S-states, $\sigma = s_0 s_1 s_2 ... s_n$, where for each $0 \leq i < n$ there exists $\pi_i \in Distr(S)$ such that $s_i \rightarrow \pi_i$ and $\pi_i(s_{i+1}) > 0$. Let $\sigma(i)$ denote the state in the i-th position. Let $|\sigma|$ be the length of σ. Let $first(\sigma) = \sigma(1)$ and $last(\sigma) = \sigma(|\sigma|)$. Let *paths*(T) denote the set of all probabilistic paths of T and *s-paths*(T) denote the sets of simple paths in T starting from any $s \in S$. A state t is reachable from other state s in T if there is $\sigma \in$ *s-paths*(T) with $s = first(\sigma)$ and $t = last(\sigma)$. Let *reach*(T,s) denote the set of all states reachable from s in T.

For any rooted FPTS(F, s), the *probability measure* $P_{F,s}$ on the σ-algebra induced by (F, s) is the unique probability measure defined such that $P_{F,s}(\sigma) = \pi_0(s_1) \cdot \pi_1(s_2) \cdot ... \pi_{n-1}(s_n)$ if (s $= s_o$) else 0. In particular, $P_{F,s}(\sigma)$ is the probability of σ in F starting from s.

Any given PTS T defines a set of *probabilistic executions*, each one obtained by iteratively scheduling one of the possible post-state distributions from each pre-state, starting from a given state $s_0 \in S$. This is the unique path leading from the start state to s.

Definition 5 *A probabilistic path of T is a FPTS,* $F = (s{-}path(T), \rightarrow_F, \mathfrak{f} \circ last)$ *where* $q \rightarrow_F \rho$ *implies* $last(q) \rightarrow_T \pi$ *with* $\rho(ps) = \pi(s)$ *for all* $s \in S$. *If in addition, for all* $q \in s - path(T)$ *such that* $|q| < i, last(q) \rightarrow_T$ *implies that* $q \rightarrow_F$, *then the rooted FPTS* (F, s_0) *is said to be a probabilistic execution fragment of length* i *of T starting from* $s_0 \in S$. *If* $i = \infty, then(F, s_0)$ *is said to be a probabilistic execution of T starting from* $s_0 \in S$.

Given a simple path $\sigma \in$ *s-paths*(T) define $\sigma^{\uparrow} \in$ *s-paths*(F)(F being a probabilistic path of T) such that $|\sigma^{\uparrow}| = |\sigma|$ and for all $0 < i \leq |\sigma|$, $\sigma^{\uparrow}(i) = \sigma(1)...\sigma(i)$. Let $\mathfrak{f} \in$ PF where PF is the set of propositional formulas closed under $\wedge$ and $\neg$. Define $\Sigma_f \triangleq \{\sigma \in s\text{-}paths(\mathrm{T})|\ last(\sigma) \models \mathfrak{f}$ and $\forall 0 < i < |\sigma|.\ \sigma(i) \models \neg\ \mathfrak{f}\}$, i.e. Σ_f is the set of all minimal paths in T that end in final condition $\mathfrak{f}$. The *minimum*(infimum) and *maximum* (supremum) *probabilities* of reaching a final condition $\mathfrak{f} \in$ PF from an initial condition $i \in$ PF in a rooted PTS(T, s_0) are defined respectively by:

$$\mathrm{P}^{inf}_{T,s_0}\ (\mathrm{i,f}) \triangleq \inf\ \{\mathrm{P}_{F,q}(\Sigma^{\uparrow}_{f})\ |\ s \in reach(\mathrm{T},\ s_0),\ s \models \mathrm{i, and\ (F,\ q)} \in execs(\mathrm{T},\ s)\}$$

$$\mathrm{P}^{sup}_{T,s_0}\ (\mathrm{i,f}) \triangleq \sup\ \{\mathrm{P}_{F,q}(\Sigma^{\uparrow}_{f})\ |\ s \in reach(\mathrm{T},\ s_0),\ s \models \mathrm{i, and\ (F,\ q)} \in execs(\mathrm{T},\ s)\}$$

where $execs(T, s)$ be the set of all probabilistic executions starting from s.

Figure 2.2 presents a probabilistic execution F of an exam PTS T. The probability to eventually pass the course(p holds) after 3 attempts ($c < 3$) is calculated as follows:
$P_{F,p} \triangleq \Sigma_{\delta \in \Sigma^{\uparrow}_{f}}\ P_{F,\delta} = 0.75 + 0.25 \cdot 1(0.75 + 0.25 \cdot 1 \cdot 0.75) = 0.9843$

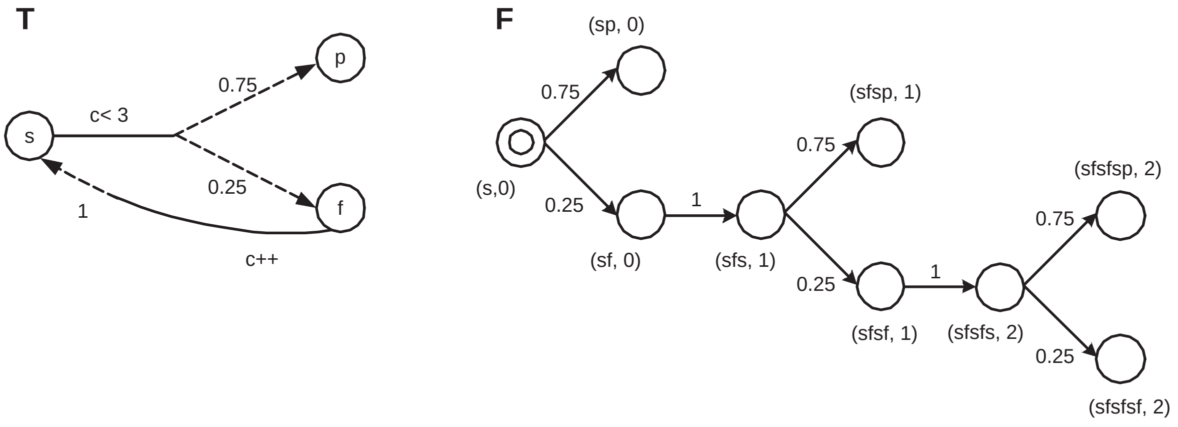


Figure 2.2: Probabilistic Execution of PTS

2.2.2 Parallel Compositions

Given two PTSs $T_1 = (S_1, \rightarrow_1, V)$ and $T_2 = (S_2, \rightarrow_2, V)$ over the same set of atomic propositions, we investigate the parallel composition of these two and hence extrapolate to the composition of several models together.

First, lets consider an asynchronous interleaving of the individual processes($|$). The parallel composition is a product composition of the states. This product structure gets huge quickly with the number of processes involved. Formally $T_1 \mid T_2 = (S_1 \times S_2, \rightarrow, V(S_1, S_2))$ where

- $(S_1 \times S_2) = \{(s_i, s_k) | s_i \in S_1 \wedge s_k \in S_2\}$
- $V(S_1, S_2) = V(S_1) \cup V(S_2)$
- $\rightarrow$ defined by the following rules:
  - If $s_1 \overset{\alpha}{\rightarrow}_1 \pi_1$ then $(s_1, s_2) \overset{\alpha}{\rightarrow} \pi_1^{s_2}$ where

$$\pi_1^{s_2}(s_1', s_2') = \begin{cases} \pi_1(s_1'); & s_2' = s_2 \\ 0; & s_2' \neq s_2 \end{cases}$$

  - If $s_2 \overset{\alpha}{\rightarrow}_2 \pi_2$ then $(s_1, s_2) \overset{\alpha}{\rightarrow} \pi_2^{s_1}$ where

$$\pi_2^{s_1}(s_1', s_2') = \begin{cases} \pi_2(s_2'); & s_1' = s_1 \\ 0; & s_1' \neq s_1 \end{cases}$$

Figure 2.3 presents the examination models, that a student takes two courses. The probability to pass both is examined on the parallel composition, depicted in figure 6.4.

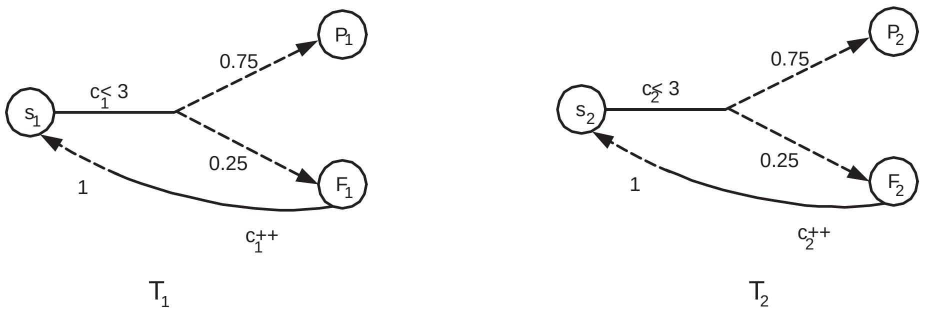


Figure 2.3: Two PTS exams T_1, T_2

As can be seen from the figure of the parallel composition, the distribution from each parallel state (s_1, s_2) to another state (s_1, p_2) is based on the distribution of the component state making the transition, that is $(s_2) \rightarrow (p_2)$, just as is defined under the rules for the parallel composition. Following the Figure 6.4 logically from the start points (s_1, s_2), a student taking the 2 courses can only

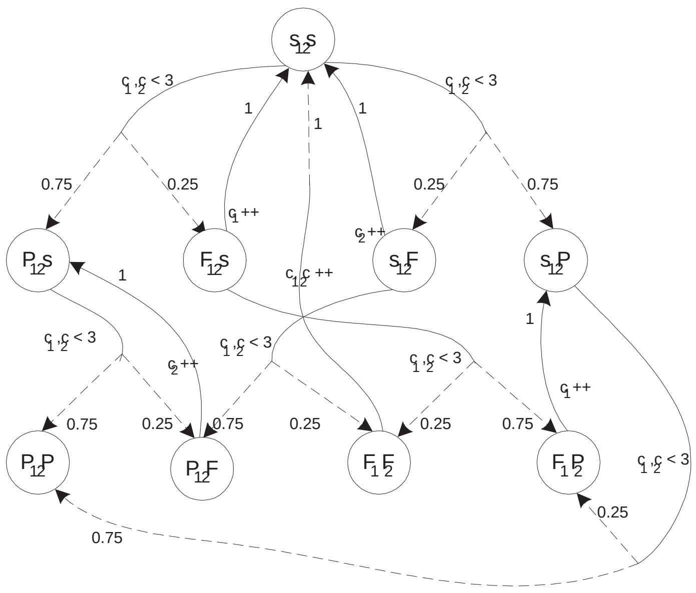


Figure 2.4: Parallel Composition of two processes, $T_1 \| T_2$

fail or pass one at a time. The first alphabet represents the first course and the second represents the second course. So one can start the exams (s_1, s_2), take the second course and fail it (s_1, F_2). From this point, one might decide to try the first course. If this is also failed you end up in (F_1, F_2) or fortunately in (P_1, F_2) if it is passed. At this point if the number of trials of the failed course c_2 is still less than 3, one can start from (P_1, s_2) and then hope to end up at (P_1, P_2).

From the above example, an interesting specification to explore could be if the probability of passing at least one of the courses taken is greater than say, 0.5. However, to carry out this exploration, one might need to consider all the execution paths incident to this parallel composition.

Secondly, we consider the synchronous parallel composition($\|$). Formally $T_1 \| T_2 = (S_1 \times S_2, \rightarrow, V(S_1, S_2))$ where

- $(S_1 \times S_2) = \{(s_i, s_k) | s_i \in S_1 \wedge s_k \in S_2\}$
- $V(S_1, S_2) = V(S_1) \cap V(S_2)$
- $\rightarrow$ defined by the following rules:
  - If $(s_1 \xrightarrow{\alpha}_1 \pi) \wedge (s_2 \xrightarrow{\alpha}_2 \beta)$ then $(s_1, s_2) \xrightarrow{\alpha} (\pi \times \beta)(s'_1, s'_2)$ where

$$(\pi \times \beta)(s'_1, s'_2) = \begin{cases} \pi_(s'_1) \cdot \beta(s'_2) & \\ 0; & (if s_1 \xrightarrow{\alpha}_1 \wedge s_2 \overset{\alpha}{\not\rightarrow}) then (s_1, s_2) \overset{\alpha}{\not\rightarrow} . \end{cases}$$

2.3 Specification Formalism

The two main approaches for specifying properties are described in this section. We state the syntax and semantics of the Probabilistic Computational Tree Logic (PCTL, [25]) and also describe a model-based specification formalism.

2.3.1 Logic-based Specification

In specifying the property to be verified in the model, the formula for the property is often stated in some logic. Branching time logics such as Computational Tree Logic(CTL) [Clarke & Emerson] allow quantification over the possible futures which leads to a formula stating eg. the existence or non-existence of an execution with a certain property. CTL distinguishes between state and path formulas. The states formulas subsume the propositional connectives and basic temporal operators of the form " a path quantifier followed by a single temporal modality" where the path quantifiers are $\forall$ and $\exists$ that range over all paths. We omit the syntax and semantics of CTL.

[Hansson & Jonsson] considered systems modelled by discrete markov chains and introduced the logics of Probabilistic Computational Tree Logic (PCTL), that can express quantitative bounds on the probability of system evolutions. This logic can thus be used to reason about the reliability and performance of systems. It is obtained by adding to the branching time logic CTL, the probabilistic operator $[]_{\sqsupseteq p}$ such that the formula $[\varphi]_{\sqsupseteq p}$ is true at a given point of the system evolution, if starting from that point, the probability that a future evolution satisfies φ is at least (most) p. PCTL allows one to express quantitative properties of probabilistic processes such as ' the system terminates with probability of at least 0.75'. It also distinguishes between state and path formulas. PCTL contains atomic propositions and operators:next-step X and until U. The operators are used in connection with an interval of probabilities.

[Bianco & Alfaro] extend the logics of PCTL to systems in which nondeterminism and probabilistic behavior co-exist. Due to the presence of nondeterminism, it is not possible, in general, to talk about the probability with which a formula is satisfied but only about the lower and upper bounds of such probability. Therefore, the formula $[\varphi]_{\sqsupseteq p}$ is true at a given point of the system evolution if the system evolution starting from that point satisfies φ with a probability bounded from below(above) by p.

The logic we could be work with, is essentially this Probabilistic Computational Tree Logic PCTL over PTSs. However in our verification process, where we only seek to establish the properties that are of the type of probabilistic reachability, these formulas only specify a quantification over path: a path or all paths. This is a fragment of PCTL as some of the assumptions we specify do not hold on all PCTL. Below we present the syntax and semantics of this PCTL over PTSs.

Syntax : We distinguish between two classes of formula: the class of state formulas(whose truth values are evaluated on the states, *state*) and the class of sequence formulas(whose truth values are evaluated on infinite sequence of states,*path*). The classes *state* and *path* are defined as follow:

$$
\begin{gathered}
\mathcal{P} \subseteq state \\
\phi, \varphi \in state \implies \phi \wedge \varphi, \neg\phi \in state \\
\phi \in path \implies A\phi, E\phi, [\phi]_{\sqsupseteq p} \in state \\
\phi \in state \implies \phi \in path \\
\phi, \varphi \in state \implies \Box\phi, \Diamond\phi, \phi\,\mathcal{U}\varphi \in path
\end{gathered}
$$

where $p \in$ [0,1] and $\sqsupseteq$ is either $\geq$ or $>$.

Semantics : For a formula $\phi \in state$, indicate with $s \models \phi$ its satisfaction on state $s \in S$, and for $\phi \in path$ indicate with $\sigma \models \phi$ its satisfaction on the infinite state sequence path σ. The semantics of the logical connectives and of the temporal operators are defined in the usual way; the semantics of A, E, $[]_{\sqsupseteq p}$ are defined as follows:

$$
\begin{gathered}
s \models A\phi \text{ iff } \forall\sigma \in s\text{-}path.\ \sigma \models \phi \\
s \models E\phi \text{ iff } \exists\sigma \in s\text{-}path.\ \sigma \models \phi \\
s \models [\phi]_{\geq p} \text{ iff } infimum(\{\sigma \in s\text{-}path | \sigma \models \phi\})_{\geq p} \\
s \models [\phi]_{\leq p} \text{ iff } supremum(\{\sigma \in s\text{-}path | \sigma \models \phi\})_{\leq p}
\end{gathered}
$$

If $s \models [\phi]_{\geq p}$, it means that regardless of the choices made in a nondeterministic state, the probability that the future evolution satisfies ϕ is at least p (and also for $s \models [\phi]_{\leq p}$). A formula $\phi \in state$ is satisfied by a rooted PTS S, written $S \models \phi$ if $s_o \models \phi$.

2.3.2 Model-based Specification

We use a transition system based specification in this thesis.Our representation of the specification is a model, which can be explicitly translated into the Probabilistic Modal Logic of Larsen et al []. We are interested in a restricted class of reachability properties, and hence interested in particular sequences of execution that lead to certain final states. These properties allow us to specify that the probability of reaching a particular final condition ϕ_f from any reachable state satisfying a given initial condition ϕ_i is smaller (or greater) than a given probability p. To minimize the complexity of the sequences possible, we later in Chapter **??**, introduce a model (Blocking PTS) which is a type of a PTS which well describes the reachable properties we work with.

2.4 Equivalences and Preorders

Simulation($\sqsubseteq$) and bisimulation relations ($\sim$) have been widely considered([31] [41]) to compare the stepwise behavior of states in transition systems. Bisimulation relations are equivalences(these are reflexive, symmetric and transitive) such that two bisimular states exhibit identical stepwise behavior. On the contrary, simulation relations are preorders (these are reflexive and transitive) on the state space such that if $s \sqsubseteq s'$ (s' simulates s) state s' can mimic all stepwise behavior of s; the converse, that is $s' \sqsubseteq s$, is not guaranteed, so state s' may perform steps that can not be matched by s. Thus if $s \sqsubseteq s'$, then every successor of s has a corresponding, related successor of s', but the reverse does not necessarily hold. Simulation can be lifted to the entire transition systems by comparing (according to $\sqsubseteq$) their initial states. Simulation relations are often used for verification purposes to show that one system correctly implements another, more abstract system.

Bisimulation relations possess the *strong preservation* property whereas simulation has *weak preservation*. Strong preservation means if $s \sim s'$, then for all formulas ϕ, it follows $s \models \phi \; iff \; s' \models \phi$. This property holds, for instance, for CTL and strong bisimulation ([13]). The use of simulation relies on the preservation of certain classes of formulas, not for all formulas (such as for $\sim$). For instance, if $s \sqsubseteq s'$ then all safety formulas ϕ, it follows that $s' \models \phi$ *implies* $s \models \phi$. However, the converse $s \not\models \phi$, cannot be used to deduce that ϕ does not hold in the simulated state s; hence the name *weak preservation*.

Simulation relations are the basis for abstraction techniques where the basic idea is to replace the large system to be verified by a small abstract model and to model check the abstract system. ([4])

In this section, we state the definitions for the bisimulation and simulation relations for a labelled transition system and then extend them to the probabilistic case.

Given a labelled transition system (LTS) as a tuple (S, A, $\rightarrow$) where S is a set of states, A is a set of actions and $\rightarrow \subseteq S \times A \times S$ the transition relation, we define $R \subseteq S \times S$ as a binary relation over S and $R^{-1} = \{\langle s', s\rangle | \langle s, s'\rangle \in R\}$ for the inverse of R.

Definition 6 *Let $\langle S, A, \rightarrow\rangle$ be a LTS and let $R \subseteq S \times S$ be a relation. Then:*

1. *R is a simulation if for every $\langle s_1, s_2\rangle \in R$ and $a \in A$, whenever $s_1 \xrightarrow{a} s_1'$, then there is a s_2' such that $s_2 \xrightarrow{a} s_2'$ and $\langle s_1', s_2' \in R\rangle$.*
2. *R is a bismulation if both R and R^{-1} are simulations.*

For any LTS there is a maximal simulation(a preorder), $\preceq$ and bisimulation(an equivalence), $\sim$. The following states a connection between $\preceq$ and $\sim$.

Theorem 1 *Let $\langle S, A, \longrightarrow\rangle$ be a LTS, with $s_1, s_2, s_3 \in S$. Then:*

1. *If $s_1 \sim s_2$ and $s_2 \preceq s_3$, then $s_1 \preceq s_3$.*
2. *If $s_1 \preceq s_2$ and $s_2 \sim s_3$, then $s_1 \preceq s_3$.*

Extending the notion of simulation to two LTSs, $T_1 = \langle S_1, A_1, \longrightarrow_1\rangle$ and $T_2 = \langle S_2, A_2, \longrightarrow_2\rangle$, we say $T_1 \preceq T_2$ if their initial states $s_1 \preceq s_2$.

With this background, we extend this equivalence and preorder to the case of Probabilistic Transition Systems. We examine the concept of Probabilistic Simulation between two PTS models and its associated states and distributions.

Definition 7 *Let S_1 and S_2 be finite sets, such that $\mathcal{R} \subseteq S_1 \times S_2$, and $\mu_1 \in \pi(S_1)$, $\mu_2 \in \pi(S_2)$. A weight function for μ_1, μ_2 with respect to $\mathcal{R}$ is a function $\delta : S_1 \times S_2 \rightarrow$ [0,1] which satisfies*

1. *For all $(s_1 \in S_1) : \Sigma_{s_2 \in S_2} \delta(s_1, s_2) = \mu_1(s_1)$*
2. *For all $(s_2 \in S_2) : \Sigma_{s_1 \in S_1} \delta(s_1, s_2) = \mu_2(s_2)$*
3. *If $\delta(s_1, s_2) > 0$ then $(s_1, s_2) \in \mathcal{R}$*

$\mathcal{R}$ an equivalence relation, is a bisimulation if and only if for all $(s_1, s_2) \in \mathcal{R}$: whenever $(s_1, s_2) \in \mathcal{R}$ and $s_1 \overset{\alpha}{\rightarrow}_1 \mu_1$ then there exists a transition $s_2 \overset{\alpha}{\rightarrow}_2 \mu_2$ and a weight function for (μ_1, μ_2) with respect to $\mathcal{R}$. Intuitively the weight function δ, show how to split the probability distributions μ_1 and μ_2 on s_1, $s_2 \in S$, see Figure 2.5, so that the relation is preserved. For simulation, the requirement that $\mathcal{R}$ is an equivalence relation is dropped (if and not if and only if). We write $\mu_1 \sqsubseteq_{\mathcal{R}} \mu_2$ if there exists a weight function between μ_1, μ_2 with respect to $\mathcal{R}$. Also $s_1 \sqsubseteq_{\mathcal{R}} s_2$ if whenever $s_1 \overset{\alpha}{\rightarrow} \mu_1$ then $s_2 \overset{\alpha}{\rightarrow} \mu_2$ with $\mu_1 \sqsubseteq_{\mathcal{R}} \mu_2$.

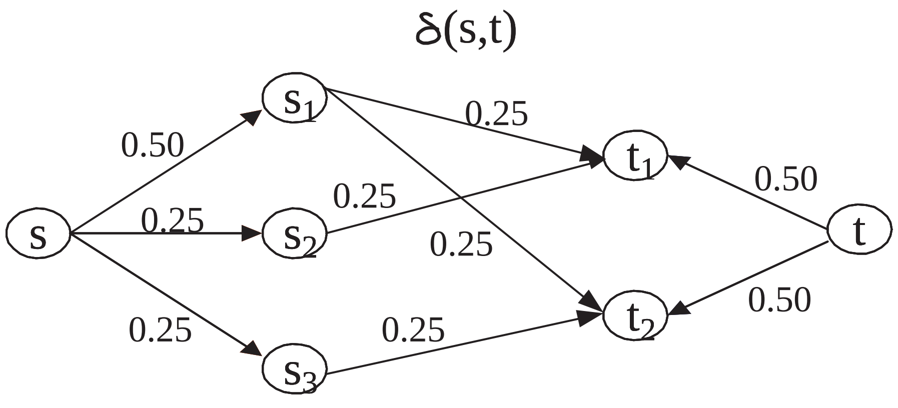


Figure 2.5: Weight Function, s $\sqsubseteq_R$ t

Definition 8 *A simulation between a rooted PTS* $T_1 = (S_1, \rightarrow_1, V_1, s_0^1)$ *and* $T_2 = (S_2, \rightarrow_2, V_2, s_0^2)$ *is a subset* $\mathcal{R}$ *of* $S_1 \times S_2$ *such that*

1. $(s_0^1, s_0^2) \in \mathcal{R}$
2. *whenever* $(s_1, s_2) \in \mathcal{R}$ *and* $s_1 \overset{\alpha}{\rightarrow}_1 \mu_1$ *then there exists a transition* $s_2 \overset{\alpha}{\rightarrow}_2 \mu_2$ *and a weight function* δ *for* (μ_1, μ_2) *with respect to* $\mathcal{R}$ *i.e.* $\mu_1 \sqsubseteq_{\mathcal{R}} \mu_2$.

We say s_1 *is simulated by* s_2*(denoted by* $s_1 \sqsubseteq s_2$*) iff there exists a simulation that contains* (s_1, s_2)*. Consequently,* $T_1 \sqsubseteq T_2$ *if the above conditions hold.*

Chapter 3

Computing The Simulation Preorder

3.1 Introduction

In using abstractions in verification, we need to establish what a good abstract is. As stated previously, we use the simulation preorder to guarantee this. In this chapter, we describe how to establish that two PTSs probabilistically simulate each other. We introduce a mathematical problem and use its solution to compute the simulation preorder relation.

By the definition of simulation over PTSs (Definition 8), finding the relation $R = S \times S'$ is the key point of determining whether one PTS simulate the other. The basic algorithm for computing the simulation relation ([16]) is as follows:

```
$R := S \times S$;
While there exists $(s, s') \in R$ with $s \not\sqsubseteq_R s'$ do
    $R := R \backslash \{(s, s')\}$

Return R.
```

Figure 3.1: The Basic Schema for computing the Simulation Preorder.

In PTSs though, the task of checking $s \sqsubseteq_R s'$ (simulation by state), extends to checking $\mu \sqsubseteq_R \mu'$ (simulation by distribution) as in the definition. In checking $\mu \sqsubseteq_R \mu'$, a network-based technique is used. The algorithm for computing the simulation relation between two PTSs, basically tests if a weight function, for distributions μ, μ' with respect to a given relation R, exists. The problem of finding a weight function is reduced to a maximum flow problem in networks.

In the next section we introduce the concept of networks and flows in them, an how the value of flow is computed in these networks.

3.2 Networks and Flow

A *network* is a tuple $\mathcal{N} = (N, E, \perp, \top, c)$ where (N, E) is a finite directed graph (i.e N is a set of nodes, E $\subseteq$ N $\times$ N a set of edges) with two specified nodes $\perp$(the source:s) and $\top$(the sink:t) and a *capacity c*. c is a function which assigns to each edge $e = (u, v) \in E$ a non-negative number $c(e)$. $c(e) : E \to R_{\geq 0}$.

A *flow function* f for $\mathcal{N}$ is a function which assigns to edge e a real number $f(e)$ such that $0 \leq f(e) \leq c(e)$ for all edges e.

Let $in(v)$ be the set of incoming edges to node v and $out(v)$the set of outgoing edges from node v. Then for each node $v \in N \setminus \{\perp, \top\}$:

$$\sum_{e \in in(v)} f(e) = \sum_{e \in out(v)} f(e)$$

Definition 9 *Flow. A flow is a function $f : E \to R$ satisfying the following*

(1) Capacity Constraint: f(u, v) $\leq$ c(u, v) , $\forall$(u, v)

(2) Skew Symmetry: f(u, v) = - f(u, v) , $\forall$(u, v) $\in$ E

(3) Flow Conservation: $\Sigma_{v \in N}$ f(u, v) = 0 ; $\forall u \in N$ - $\{s, t\}$

The *value* of a flow f is $|f| = \Sigma_{v \in N} f(s, v)$.

The *excess* of a node (v) is

$$excess(v) = \Sigma_{e \in in(v)} f(e) \text{ - } \Sigma_{e \in out(v)} f(e)$$

The *maximum flow* in $\mathcal{N}$ is the suprenum (maximum) of the value of flow in the network where f is a flow function in $\mathcal{N}$.

3.3 Maximum Flow Problem

Given a network $(N, E, \perp, \top, c)$, find a flow of maximum value from source($\perp$) to sink($\top$) i.e. determine a flow f for which $|f|$ is maximum.

Finding this maximum flow in a network is achieved by adjusting the flow and capacities on the edges until they are stable. From the definition of the flow function, we assign flows to the edges based on their capacities. We present this on the edges as *capacity/flow*, Figure 3.2. The flows into a node(except for the source and sink nodes) must be equal to the flows out of it. Consequently, the *excess* of nodes is 0.

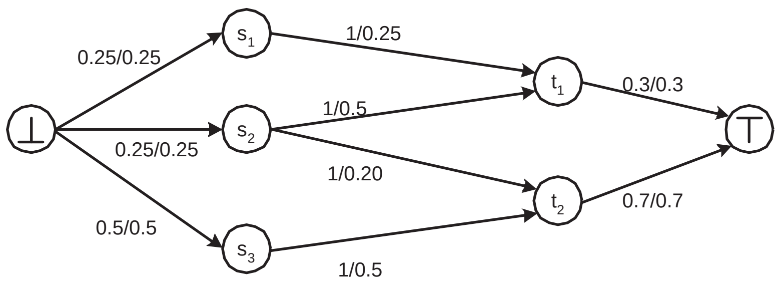


Figure 3.2: A Network with Maximum Flow of 1

Residual Networks

Let $\mathcal{N}$ be a network with a flow f. For any $(u, v) \in E$, the residual capacity of (u, v) is $c_f(u, v) = c(u, v) - f(u, v)$. The *residual graph* of $G = (N, E)$ induced by f is

$$G_f = (N, E_f)$$

where

$$E_f = \{(u, v) \in N \mid c_f(u, v) > 0\}$$

The flow f also gives rise to the residual flow network $\mathcal{N}_f = (G, \bar{c_f}, s, t)$ where $\bar{c_f}(u, v) = c_f(u, v)$ for $(u, v) \in E_f$ and 0 otherwise(i.e. for $(u, v) \in E - E_f$).

Given a graph $G = (N, E)$ and a *flow* f. An *augmenting path* π is a simple path from s to t in the residual graph, G_f, induced by *flow* f. Every edge in G_f has positive capacity. The maximum amount of net flow that can flow along edges of π is called the *residual capacity of* π:

$$c_f(\pi) = min\{c_f(u, v) \mid (u, v) \text{ is on } \pi\}$$

For every edge $e = (u, v)$ in G there are up to two edges e' and e'' in G_f

1. If $cap(e) < f(e)$, $e' = (u, v) \in G_f, r(e') = cap(e) - f(e)$
2. If $f(e) > 0$, $e'' = (v, u) \in G_f, r(e'') = f(e)$
3. If $c(e) = f(e)$, $e'' = (v, u) \in G_f, r(e'') = c(e)$

Lemma Fix $\mathcal{F}= (G, c, s, t)$, with flow f, and augmenting path π in G_f , define

$$f_\pi(u,v) = \begin{cases} f(e) + c_f; & if e' \in \pi \\ f(e) - c_f; & if e'' \in \pi \\ 0; & otherwise \end{cases}$$

The generic algorithms for calculating maximum flow are based on general graphs. Some examples are the Ford-Fulkerson Algorithm, Dinic's Algorithm, and the First-In First-Out($FIFO$) Preorder Push Algorithm. Modifications have been made to adapt these algorithms to bipartite graphs, which is more useful in our case. When the probabilistic simulation problem is reduced to the maximum flow problem, a network $\mathcal{N} = ((N,E),\bot,\top,c)$ is established. $G = (N,E)$ is actually a bipartite graph, written $G = (X \cup Y, E)$. We implement an improved version of the Ford-Fulkerson Algorithm for bipartite graphs. This improved algorithm performs at $O(p^4)$ where p = max $\{\mid X \mid, \mid Y \mid\}$.

The Ford-Fulkerson Algorithm

Given a network (G,s,t), with source and sink nodes, s and t respectively,

1. Initialize flow f to 0
2. *While* there exists an augmenting path π in G_f
   Do augment flow f along π
3. Return f

Given the graph G with special nodes s and t as source and sink nodes, respectively, the algorithm starts with zero flows $f(e) = 0$ for all edges e. It then constructs the residual network G_f. In this residual network, it checks whether t can be reached from s. If there is an augmenting path, then this is possible. If not it stops else $flow$ is adjusted along the augmenting path and it iterates. The value of $flow$ obtained when there is no augmenting path, has been found to be the Maximum Flow of the network.

3.4 Probabilistic Simulation and Maximum Flow

We now show how the problem of computing the probabilistic simulation($\mu \sqsubseteq_R \mu'$), is reduced to that of The Maximum Flow Problem.

For each transition $s \overset{\alpha}{\rightarrow} \mu$, let $Child_{s,\alpha}(\mu) \subseteq S$ is a set of states, whose elements are distributed by μ, for instance, in Figure 3.4 $Child_{s_0,\alpha}(\mu) = (s_1, s_2, s_3)$. Given the preorder relation $R \subseteq S \times S'$, $s \overset{\alpha}{\rightarrow} \mu$ and $s' \overset{\alpha}{\rightarrow}_{'} \mu'$, where $\mu \in Dist(S)$, $\mu' \in Dist(S')$. Choose $\perp, \top$ such that $\perp = \ s, \top = \ s'$. We derive a network $\mathcal{N}$ $(N, E, \perp, \top, c)$ or $\mathcal{N}(\mu, \mu', R)$ such that

$N = \{\perp, \top\} \cup \{Child_{s,\alpha}(\mu) \cup Child_{s',\alpha}(\mu')\}$

$E = \{(s, s') : (s, s') \in R\} \cup \{(\perp, s) : s \in Child_{s,\alpha}(\mu)\} \cup \{(s', \top) : s' \in Child_{s',\alpha}(\mu')\}$

$c(\perp, s) = \mu(s), c(s', \top) = \mu'(s), c(s, t') = 1$

Lemma: The following are equivalent

1. There exists a weight function δ for (μ, μ') with respect to R
2. The maximum flow in $\mathcal{N}$ (μ, μ', R) is 1.

The algorithm that computes the maximum flow in the probabilistic simulation induced network $\mathcal{N}$, is given in Figure 3.3.

```
Input : A nonempty, finite set $S$, distribution $\mu, \mu' \in Distr(S)$and $R \subseteq S \times S$
Output : If $\mu \sqsubseteq_R \mu' \in$ then "Yes"
else "No"
Method :
    Construct the network $\mathcal{N}(\mu, \mu', R)$;
    Compute the maximum flow F in $\mathcal{N}(\mu, \mu', R)$;
        If $F < 1$ then return then "No"
        else "Yes"
```

Figure 3.3: The Test for $\mu \sqsubseteq_R \mu'$.

As an example consider the PTSs in Figure 3.4. In order to check whether μ' simulates μ, with relation $R = \{(s_1, t_1), (s_2, t_1), (s_2, t_2), (s_3, t_2), ...\}$ we can establish a network for each state, as shown in Figure 3.2.

Example: Applying the algorithm for computing the probabilistic simulation relation R for two given PTSs, T and T' in Figure 3.4. We start with the relation R containing the pairs:

- (s_0, t_0), (s_0, t_3), (s_3, t_2), (s_5, t_3),
- (s_i, t_j), where $(i = 1, 2, 4, 6, ..., 9)$ and $(j = 0, ..., 4)$

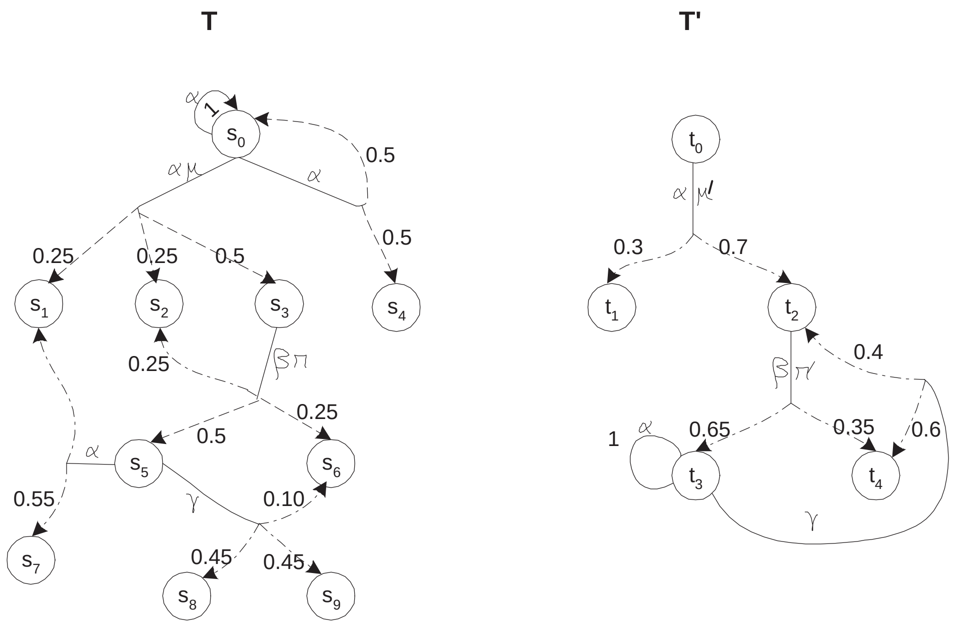


Figure 3.4: An example of a PTS

Intuitively, there are some pairs which are not in the initial R, such as (s_5, t_0) because the action set over $(s_5) \nsubseteq$ the action set over(t_0), $(\{\gamma\} \notin \{\alpha\})$.
The pair (s_0, t_3), (s_3, t_2) are also removed from the set R during the investigation. For (s_0, t_3), $Distr(s_0, \alpha) = \{\mu_{s_0}, \mu_{s_4}, \mu\}$ and $Distr(t_3, \alpha) = \{\mu_{t_3}\}$, and as $\mu \nsubseteq_R \mu_{t_3}$ then $s_0 \nsubseteq_R t_3$. For (s_3, t_2) the computed maximum flow is 0.85, which implies $s_3 \nsubseteq_R t_2$. Although the pair (s_0, t_0) is still in R in the initial investigation as $Distr(s_0, \alpha) = \{\mu_{s_0}, \mu_{s_4}, \mu\}$ and $Distr(t_0, \alpha) = \{\mu'\}$, it is later removed, because after the pair (s_3, t_2) is removed from R, its maximum flow of (μ, μ') is adjusted and is less than 1. Finally, we get the relation R containing the pairs:

- (s_5, t_3),
- (s_i, t_j), where $(i = 1, 2, 4, 6, ..., 9)$ and $(j = 0, ..., 4)$

Putting it all together, we now present the algorithm that computes the probabilistic simulation relation between two given PTSs.

Initialization:

$R := \{(s,s') \in S \times S : acts(s) \subseteq act(s')\}$

For all $(s,s') \in R$ and $s \to \mu$ do $Sim_{(s,\alpha,\mu)}(s') := Steps_\alpha(s'))$

Iteration:

```
    **Repeat:** ;
        $R_{old} := R; R := 0$
        For all $(s,s') \in R_{old}$ do
          • $sim := true$;
          • For all $s \to \mu$ do
              Repeat:
                choose some $\mu' \in Sim_{(s,\alpha,\mu)}(s')$;
                  If $\mu \sqsubseteq_R \mu'$ then remove $\mu'$ from $Sim_{(s,\alpha,\mu)}(s')$;
              until $Sim_{(s,\alpha,\mu)}(s') = 0$ or $\mu \sqsubseteq_R \mu'$;
              If $Sim_{(s,\alpha,\mu)}(s') = 0$ then $sim := false$;
          • If $\neg sim$ then $R := R \cup \{(s,s')\}$;
    until $R_{old} = R$;
```

Output: Return R.

Figure 3.5: Basis algorithm for computing the Simulation Preorder.

Chapter 4

General Methods for Compositionality & Abstraction

Frege's Principle of compositionality: The meaning of the whole is a function of the meaning of the parts.

Abstraction is one of the most useful ways to fight the state explosion problem. They should however preserve the properties of interest such that properties that hold for the abstract system should hold for the concrete model. Model abstraction reduces the number of states necessary to perform formal verification while maintaining the functionality of the original model with respect to the specification to be verified . As a result model abstraction enables large designs to be formally verified. The resulting abstract models can replace the original model for formal verification provided that each of the abstractions is homomorphic to the corresponding part of the original model that it replaces with respect to the specification to be verified([34])

Model Checks $M \models \phi$, can be abstracted by simplifying the model M ([17]), the property, ϕ ([26]), or the satisfaction relation, $\models$.

Model partitioning takes a portion of a model and replaces it with an abstract model. E.g. if a portion of a model does not affect (i.e. is independent from) the rest of the model with respect to the properties to be verified, it may be advantageous to abstract that portion of the model away.

4.1 Compositional Abstraction

In this section we describe work that uses the concepts of compositionality and abstraction to break down and localize abstraction to the individual processes of a system that synchronize for the total behavior of the system. In our previous work, we used this approach to help minimize the state space explored during model checking and also developed a tool called CAPS (acronymed from Compositional Abstraction by Probabilistic Simulation).

Given the model checking problem:

$$C_1||C_2||C_3||...||C_n \models \xi \tag{4.1}$$

where $C_1, C_2, C_3, ..., C_n$ are components of an asynchronous parallel system. We replace this equation with

$$A_1||A_2||A_3||...||A_k \models \xi \tag{4.2}$$

where

$$\begin{gathered} A_1 \cdots A_k \text{ are abstract components satisfying} \\ C_1||...||C_{i_1} \sqsubseteq A_1 \\ C_{i_1+1}||...||C_{i_2} \sqsubseteq A_2 \\ \vdots \\ C_{i_{k-1}+1}||...||Cn \sqsubseteq A_k \end{gathered}$$

This result was upheld provided that $\sqsubseteq$ satisfied the following properties :

$$\begin{gathered} (\textbf{Precongruence})\ C \sqsubseteq A \implies C||R \sqsubseteq A||R \\ (\textbf{Property Preservation})\ (A \models \xi \wedge C \sqsubseteq A) \implies C \models \xi \end{gathered}$$

We used the probabilistic simulation preorder [44],[29] as the relation which must hold between the original component and the abstraction. We chose the probabilistic extension of the simulation preorder, and not an equivalence relation such as bisimulation, because it permits a smaller model being obtained or in the worst case an equal model. Bisimulation forces a strict equivalence which will often not be of much help in an abstraction. This method of abstraction to minimize the model can only be used in the model checking, if the probabilistic simulation preorder has been established. If for all the components of a model, an abstraction can be found for each component, in such a way that each component is probabilistically simulated by its abstraction, we can then substitute these abstractions for the components and proceed with model checking. Consequently, we avoid the explicit construction and exploration of the state space of the original model with significant savings on time and space.

4.2 Partitioning

Model partitioning techniques reduce the state space by grouping several states into the same abstract state (i.e. partition) and by removing parts of the model not related to the specification to be verified. The abstractions considered are usually obtained by successive refinement , starting from an initial coarse partitioning of the state space, derived from the property under study. If the analysis of this abstract PTS allows to conclude that the property is satisfied by the concrete PTS, the verification process is finished. Otherwise a partition refinement step is performed in order to obtain more precise information. The process is iterated up to success or until all classes of the partition are stable. If the latter occurs, it can conclude that the property is false and extract a counter-example path, ([20]). In Section 2.2.1, we described how the maximum and minimum probabilities of reaching a final condition(ϕ_f or state) from an initial condition (ϕ_i or state) are obtained. This section is based on work done by Pedro D'Argenio et al ([20]) and their approach in obtaining good abstractions by partitioning.

Lets denote the sets of states satisfying ϕ_i and ϕ_f by I and F respectively. If $s \in F$ then $F^{inf}(f)(s) = F^{sup}(f)(s) = 1$, that is the state s is in (satisfies) the final condition. If $s \notin F$ then

$F^{inf}(f)(s) = min_{(s \to \pi)} \sum_{s' \in S} \pi(s') \cdot f(s')$ and
$F^{sup}(f)(s) = max_{(s \to \pi)} \sum_{s' \in S} \pi(s') \cdot f(s')$

In ([20]), the equations are transformed into a linear optimization problem which is solved by linear programming.

A partition is induced by an equivalent relation. We now define an equivalence relation based on simulation.

Definition 10 *Let $(S, \longrightarrow, V)$ be a probabilistic transition system. Let $C \subseteq S \times S$ be a relation on states defining a discrimination criterion. R is a C- probabilistic simulation if, whenever sRt,*

1. *$(s,t) \in C$, and*
2. *$if s \longrightarrow \pi$, and $t \longrightarrow \rho$ and there exist $\delta \in Distr(S \times S)$ such that for all $s, t \in S$,*
   - **(a)** *$\pi(s) = \delta(s, S)$*
   - **(b)** *$\rho(t) = \delta(S, t)$, and*
   - **(c)** *sRt whenever $\delta(s,t) > 0$.*

We say that t C-simulates s, notation $s \preceq_C t$ if there is a C-simulation R such that sRt.
Our interest is to check when a PTS reaches a goal ϕ_f starting from any state satisfying some initial condition i. Let C_{ϕ_i,ϕ_f} be the discriminating condition defined by
$(s,t) \in C_{\phi_i,\phi_f} \implies (s \models \phi_f \Leftrightarrow t \models \phi_f)$ and $(s \models \phi_i \Leftrightarrow t \models \phi_i)$.

Our main purpose is to answer the question whether the probability of eventually reaching the final condition f from any state satisfying a given initial condition i is smaller than a given value $p \in [0,1]$. C is an equivalence relation. The next theorem states that if a PTS T_α satisfies this property, and another PTS T is C-simulate by T_α, then T also satisfies the property.

Theorem 2 *Let (T_1, s_0^1) and (T_2, s_0^2) be two rooted PTSs, such that none of them has a sink node and let C_{ϕ_i,ϕ_f} be the discriminating condition as defined. Then*

1. *$(T_1, s_0^1) \preceq_C (T_2, s_0^2)$ implies $P^{sup}_{T_1,s_0^1}(\phi_i,\phi_f) \leq P^{sup}_{T_2,s_0^2}(\phi_i,\phi_f)$ and*
2. *$P^{inf}_{T_1,s_0^1}(\phi_i,\phi_f) \geq P^{inf}_{T_2,s_0^2}(\phi_i,\phi_f)$.*

A PTS can be abstracted by partitioning its state space, and any such partition will induce an abstract PTS which should simulate the original (concrete) one. See Theorem 4. Consequently, the abstract model will satisfy the same reachability properties as the original model. The minimum and maximum properties is preserved by the abstract system, and establish its limits (bounds).

Theorem 3 *Let $A = \{A_1, \cdots, A_k\}$ be a partitioning of the finite set of states S of a PTS T, then the following holds:*

1. *$A_i \subseteq S$*
2. *$A_i \bigcap A_j = \oslash$ $(i \neq j)$*
3. *$\bigcup_{i=1\ldots k}\{A_i\} = S$*

Definition 11 *Let T be a PTS and $A = \{A_i, \cdots, A_k\}$ be a partitioning of the states of T. Then the partitioned PTS $T/A = (A, \rightarrow_A, f_A)$ where*

1. *A is its finite set of states,*
2. *$\rightarrow_A$ transitions: $A \xrightarrow{a} \Pi$ iff there exists $s \in A$ with $s \xrightarrow{a} \pi$ and $\Pi(A') = \sum_{s' \in A'} \pi(s')$*

3. $f_A = \bigwedge_{s \in A} f(s)$.

For a rooted PTS (T, s_0), the partitioned PTS $(T, s_0)/A = (T/A, A_i)$ provided $s_0 \in A_i \in A$. This means that the partition that contains the initial state of the PTS T, becomes the initial state (partition) of the partitioned PTS.

Theorem 4 *For a PTS T and its partitioned self* (T/A), $T \sqsubseteq T/A$.

As an example, consider a coin being tossed to play out the throw of a die. Refer to Figure 4.1. To throw a "*two*", the toss sequence of the coin will be a *head*, *tail* and finally a *head*. An interesting property to verify in this model will be finding out the probability of throwing a particular number, say a "*six*". i.e. the probability of reaching a "*six*" in three successive coin tosses, (the sequence $Tail, Tail, Head$). By PCTL logic, we obtain the minimum and maximum probabilities for the final condition $\mathfrak{f} \in$ PF (propositional formulas) from an initial condition $i \in$ PF, as *a six* from *coin* (s_0, the initial state), for the die will be obtained as

$$\mathrm{P}^{inf}_{D,s_0}\ (s_0, \text{"}six\text{"}); \qquad \mathrm{P}^{sup}_{D,s_0}\ (s_0, \text{"}six\text{"});$$

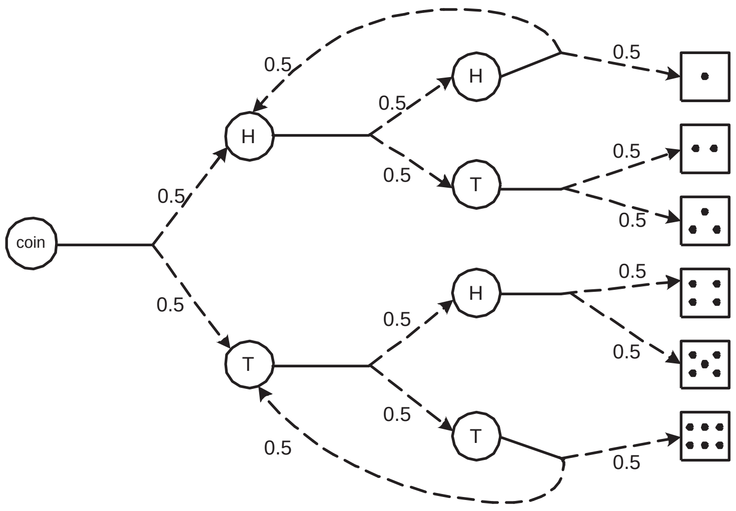


Figure 4.1: A fair coin toss as a die thrown

We obtain a coarse partition on the state of the die (D), based on the reachability property of interest, obtaining a "*six*". In Figure 4.3, we present an intuitive abstraction which is an illustration of the partitioned PTS, where the states are infact independent partitions. This is a significant reduction in the states, from 13 to 6 states, to be exponentially explored. We first show which states is put in one partition, by enclosing these in the dotted square box.

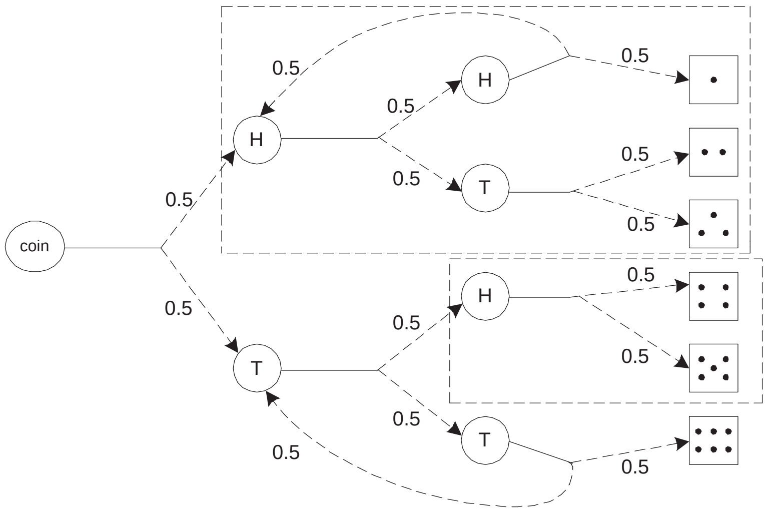


Figure 4.2: Partitioning the dice model

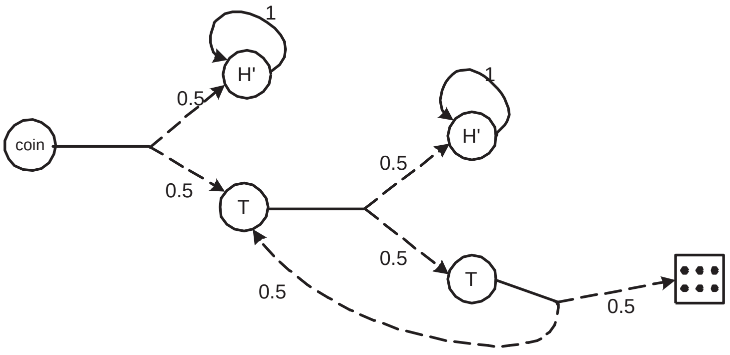


Figure 4.3: An abstraction of the coin toss

4.3 The Quotient Method

In this section we discuss the quotient technique, first introduced by Kim G. Larsen in his PhD thesis [38], which is a promising technique for avoiding the state explosion problem in model checking. It has been studied within the last decade and has been proven to be successful for finite-state systems and real-time systems [3, 35]. Our aim is to extend the method for probabilistic systems. Consider the following model checking problem involving a system with n processes in parallel:

$$A_1 \rceil\lceil ... \rceil\lceil A_n \vdash \varphi \tag{4.3}$$

where parameters of components, parallel composition ($\rceil\lceil$), specification formalism and the satisfaction relation in (4.3) may be instantiated as follows:

1. Component Type

   - Finite State System $FS = (S, \rightarrow)$, where S is a finite set of states and $\rightarrow \subseteq S \times Act \times S$ is a transition relation.
   - Timed Automata $TA = (L, l_0, E, Label, C, clocks, guards, inv)$, where L is a finite set of locations with an initial location l_0; $E \subseteq L \times L$ is a set of edges; $Label : L \rightarrow 2^{AP}$ a function that assigns to each location, a set of $Label(l)$ of atomic propositions; C is a finite set of clocks; $clocks : E \rightarrow 2^c$, a function that assigns a set of clocks, $clocks(e)$ to each edge; $guard : E \rightarrow \Psi(C)$ a function that labels each edge, with a clock constraint $guard(e)$ and $inv : L \rightarrow \Psi(C)$ a function that assigns to each location an invariant.
   - Probabilistic Transition System $PTS = (S, \rightarrow, V)$, where S is a finite set of states, $\rightarrow \subseteq S \times Act \times Dist(S)$, is a finite transition relation, where $Dist(S)$ is distribution over states S,

2. Parallel Composition

   - Interleaving

$$\frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}, \quad \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$$

   - Synchronization

$$\frac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\alpha} Q'}{P||Q \xrightarrow{\alpha} P'||Q'}$$

   - Mixed Synchronization

$$\frac{P \xrightarrow{\alpha} P'}{P|||Q \xrightarrow{\alpha} P'|||Q} \quad , \quad \frac{Q \xrightarrow{\alpha} Q'}{P|||Q \xrightarrow{\alpha} P|||Q'} \text{ and}$$

$$\frac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\alpha} Q'}{P|||Q \xrightarrow{\alpha} P'|||Q'}$$

   where P, Q, P', Q' can be one kind of the components in (1); $(|), (||)$ and $(|||)$ respectively present an interleaving, a synchronous and a mixed synchronous parallel composition.

3. Specification Formalism
    - Logic, where the specification formalism is presented in terms of a logic, such as Hennessy Milner Logic, CTL, TCTL, PCTL, normally denoted by $\phi, \varphi, \ldots$
    - Model, where the specification is presented by a model, such as Finite State, Timed Automata, Markov Chain or Labelled Transition Systems, normally denoted by A, P, T.
4. Satisfactionality Relation ($\vdash$)
    - For the logic, it is a logic satisfaction, denoted by ($\models$).
    - For the model, it could be a simulation preorder ($\preceq$, $\succeq$), probabilistic simulation preorder ($\sqsubseteq$, $\sqsupseteq$) or a bisimulation equivalance ($\sim$).

In the model checking equation, (4.3), we wish to verify that the parallel composition of those systems satisfies φ without having to construct the complete state space of $A_1 \rceil\lceil ... \rceil\lceil A_n$. We will avoid this complete construction by removing the component A_i one by one from the parallel system, while simultaneously transforming the formula accordingly. Thus, when factoring out the component A_n we will transform the formula φ into the quotient formula φ/A_n such that:

$$(A_1 \rceil\lceil ... \rceil\lceil A_n) \vdash \varphi \text{ if and only if } (A_1 \rceil\lceil ... \rceil\lceil A_{n-1}) \vdash \varphi/A_n$$

However, while repeatedly applying quotienting, another problem arises: the state explosion now occurs in the size of the quotient formula. Therefore the idea behind the Quotient Technique is that each quotienting should be followed by a simplification heuristic, such as minimization method, which will be discussed in the next chapter. We consequently obtain a combined process of quotienting and minimizing as:

$$(A_1 \rceil\lceil ... \rceil\lceil A_n) \vdash \varphi \text{ if and only if } (A_1 \rceil\lceil ... \rceil\lceil A_{n-1}) \vdash (\varphi/A_n)^s$$

By repeatedly applying quotient and simplifying the problem we finally achieve the following clause:

$$(A_1 \rceil\lceil ... \rceil\lceil A_n) \vdash \varphi \text{ if and only if } \circledast \vdash ((((\varphi/A_n)^s)/A_{n-1})^s/\ldots/A_1)^s$$

where $\circledast$ is the unit with respect to parallel composition.
In our thesis, we are interested in applying the quotient method for a model specification such as finite-state or PTS specification rather than a logical specification. For this reason, we henceforth investigate the quotient technique for the following concrete model checking equation:

$$A_1 \rceil\lceil ... \rceil\lceil A_n \vdash B$$

where,

- Components A_i, B are FSs, and the satisfaction relation ($\vdash$) is simulation preorder ($\preceq, \ \succeq$) in case of finite-state systems or,

- Components A_i, B are PTSs, and the satisfaction relation ($\vdash$) is probabilistic simulation preorder ($\sqsubseteq, \ \sqsupseteq$) in case of probabilistic systems.

We now restate that the purpose of the quotient technique is to try to avoid the state-explosion problem in parallel systems by factoring out components, one at a time while simultaneously transforming the specification accordingly for the whole system and thereafter applying simplification heuristic repeatedly for each quotienting. In this thesis we discuss the quotient technique for finite-state systems by adapting the equation solving method proposed by Larsen and Xinxin [47] and extend the methods for the probabilistic labelled transition systems.

4.3.1 Special Subclasses

In this project we focus on applying the quotient technique for special subclasses of model checking. The aim is to answer the question when a parallel composition simulates ($\succeq, \ \sqsupseteq$) or is simulated ($\preceq, \ \sqsubseteq$) by a model specification without having to construct the complete state space by using the quotient technique in finite-state systems and probabilistic systems. Consider the concrete model checking (inequation):

$$A_1 \rceil\lceil ... \rceil\lceil A_n \bowtie B$$

where $\bowtie$ is either ($\succeq, \ \sqsupseteq$) or ($\preceq, \ \sqsubseteq$).
Let $X_n = B$, by applying the quotient technique we might obtain the following set of equivalent equations:

$$\begin{aligned}
& A_1 \rceil\lceil ... \rceil\lceil A_n && \bowtie X_n \\
\textbf{iff } & A_1 \rceil\lceil ... \rceil\lceil A_{n-1} && \bowtie X_n / A_n = X_{n-1} \\
& \vdots && \\
\textbf{iff } & A_1 && \bowtie X_2 / A_2 = X_1
\end{aligned}$$

iff $\circledast \quad \bowtie X_1/A_1 = X_0$

where $\circledast$ is the unit with respect to parallel composition.
Now clearly, if for each $i \in 1..n$ the quotient $(X_i/A_i) = X_{i-1}$ is the smallest/largest, respectively for $(\succeq, \sqsupseteq)$ and $(\preceq, \sqsubseteq)$, with respect to (probabilistic) simulation preorder, we have succeeded in "quotient" solving the problem. More precisely, we give the following:

Corollary 1 *Given component A and specification B, the existence of quotients is a construct B/A such that for all X the following holds:*

$$A \rceil\lceil X \bowtie B \textit{ iff } X \bowtie B/A$$

Intuitively, the main problem now is that given a model specification B and a component A we wish to find the largest/smallest X such that $A\rceil\lceil X \bowtie B$. In the next sections we discuss the quotient technique applied to finite-state systems. The quotient technique for probabilistic systems is discussed in Chapter 7

Chapter 5

Methods for Finite State

In this chapter, we examine two main methods for finite states and in subsequent chapters extend them to the probabilistic transition system.

5.1 Minimization

In this section, we introduce Orna Grumbergs work on Simulation based Minimization ([16]). By this, we aim at being able to reduce (minimize) the structure of a model, as an individual component or as a result of some parallel composition.

In her paper, Grumberg presents a minimization algorithm which receives a Kripke structure, M and returns the smallest structure which is simulation equivalent to M. The reduced structure is obtained is based on simulation equivalence. Although bisimulation equivalence has the advantage of preserving more expressive logics, it requires the abstract structure to be too similar to the original thus allowing less powerful reductions.

5.1.1 Definitions and Theorems

Definition 12 Kripke Structure *M over atomic proposition(AP), is a four tuple $M = (S, s_0, R, L)$ where*

- *S is a finite set of states;*
- *$s_0 \in S$ is the initial state;*
- *$R \subseteq S \times S$ is the transition relation that must be total i.e., for every state $s \in S$ there is a state $s' \in S$ such that $R(s, s')$ and*

- *$L : S \rightarrow 2^{AP}$ is a function that labels each state with the set of atomic propositions true in that state.*

Definition 13 *The size $|M|$ of a Kripke structure M is a pair $(|S|, |R|)$. We say that $|M| \leq |M'|$ if $|S| \leq |S'|$ or $|S| = |S'|$ and $|R| \leq |R'|$*

In Figure 5.1, $|M^\star| \leq |M|$ because although $|S^\star| = |S|$, $|R^\star| \leq |R|$.

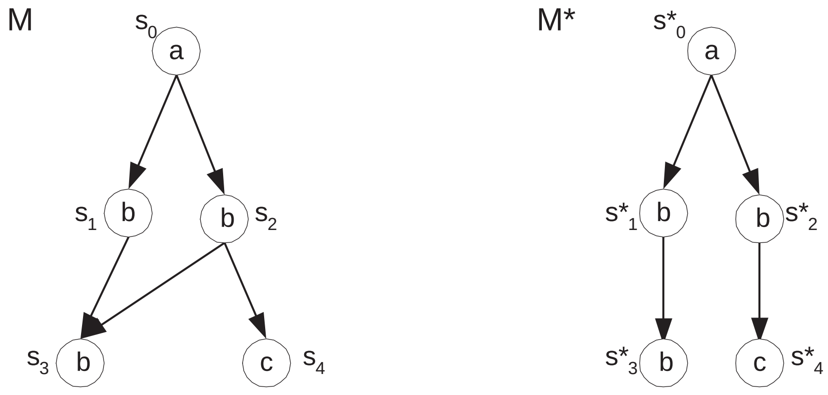


Figure 5.1: An example of a Kripke structure

Definition 14 *Given two structures $M = (S, s_0, R, L)$ and $M' = (S', s'_0, R', L')$ over AP, a relation $H \subseteq S \times S'$ is a simulation relation over $M \times M'$ iff the following holds*

1. $(s_0, s'_0) \in H$
2. $\forall (s, s') \in H, L(s) = L'(s')$ *and*
   $\forall t[(s, t) \in R \rightarrow \exists t'[(s', t') \in R' \wedge (t, t') \in H]]$.

We say that M' simulates M ($M \preceq M'$), see Figure 5.2, if there exists a simulation relation H over $M \times M'$.

Definition 15 *Given two Kripke structures M, M', we say that M is simulation equivalent to M' iff $M \preceq M'$ and $M' \preceq M$.*

A *simulation relation H* over $M \times M'$ is *maximal* iff for all simulation relations H' over $M \times M'$, $H' \subseteq H$.

Let M be a Kripke structure. The *maximal simulation relation* over $M \times M$ always exists and is denoted by H_M.

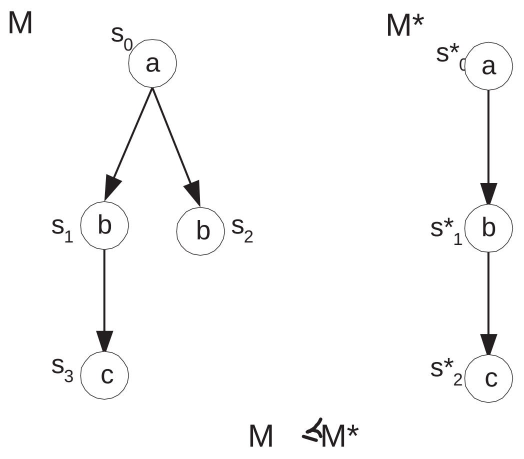


Figure 5.2: Simulation of two Kripke structures

Definition 16 *Two states $s_1, s_2 \in M$ are simulation equivalent iff $(s_1, s_2) \in H_M$ and $(s_2, s_1) \in H_M$.*

Definition 17 *A state s_1 is a little brother of a state s_2 iff there exists a state s_3 such that:*

- $(s_3, s_2) \in R$ *and* $(s_3, s_1) \in R$
- $(s_1, s_2) \in H_M$ *and* $(s_2, s_1) \notin H_M$

Definition 18 *A Kripke Structure is reduced if:*

1. *There are no simulation equivalent states in M*
2. *There are no states s_1, s_2 such that s_1 is a little brother of s_2*
3. *All states in M are reachable from s_0*

Theorem 5 *Let M be a non-reduced Kripke structure, then there exists a reduced Kripke structure M' such that M, M' are simulation equivalent and $|M'| < |M|$.*

5.1.2 The Minimizing Algorithm

The algorithm receives a Kripke structure M, and computes a reduced Kripke structure M' which is simulation equivalent to M and $|M'| \leq |M|$.

The algorithm consists of three steps. First, a quotient structure is constructed to eliminate equivalent states. The resulting model is simulation equivalent to M but may not be reduced. The next step disconnects little brothers and the final step removes all unreachable states.

We state the Minimizing Algorithm, Figure 5.3 and then expand on each phase of it.

STEP 1 Compute the $\forall - \textit{quotient}$ structure M_q of M and the maximal simulation relation H_M over $M_q \times M_q$

STEP 2 $R' = R_q - \{(s_1, s_2) | \exists s_3 : (s_1, s_3) \in R_q \wedge (s_2, s_3) \in H_M\}$

STEP 3 Remove all unreachable states

Figure 5.3: The Minimizing Algorithm.

STEP1

In order to compute a simulation equivalent structure that contains no equivalent states, we compute the $\forall - \textit{quotient structure}$ with respect to the simulation equivalence relation.

Definition 19 *The $\forall - \textit{quotient structure}$ $M_q =< S_q, R_q, s_{0q}, L_q >$ of M is defined as follows:*

- *S_q is the set of the equivalent classes of the simulation equivalence.*
- $R_q = \{(\alpha_1, \alpha_2) | \forall s_1 \in \alpha_1, \exists s_2 \in \alpha_2 . (s_1, s_2) \in R\}$
- $s_{0q} = [s_{0q}]$
- $L_q([s]) = L(s)$.

where $[s]$ is the equivalence class which includes s.

The transitions in M_q are $\forall-$transitions, in which there is a transition between two equivalence classes iff every state of the one has a successor in the other.

The output from this step is a structure with no equivalent states.

STEP2

The algorithm(See Figure 5.4) in this step iteratively disconnects little brothers in the output from STEP 1.

```
change := true
while (change = true) do
    Compute the maximal simulation relation $H_M$
    change := false
    if there are $s_1, s_2, s_3 \in S$ such that $s_1$ is a little brother of $s_2$ and $s_3$ is the
        father of both $s_1$ and $s_2$
        then change := true $R = R \setminus \{(s_3, s_1)\}$
    endif
end
```

Figure 5.4: The Disconnecting Algorithm.

The output from this step has the same number of states as the input but less transitions.

STEP3

This step removes all unreachable states from the initial state, from the structure.

An example of the algorithm is illustrated in Figure 5.5. At the first step of the algorithm, the maximal simulation relation on M (Side 1 of the Figure), and the equivalent classes are:

H_M	Equivalent Classes
{(11, 2),	{{1},
(11, 3),	{11},
(4, 5),	{4},
(6, 5),	{5},
(2, 3), (3, 2),	{2, 3},
(7, 8), (8, 7),	{7, 8},
(9, 10), (10, 9)}	{9, 10}}.

The equivalent classes, now are the states of the $\forall - structure$, Part 2. The maximal simulation relation H_M of this new structure is now { ({11},{2, 3}), ({4}, {5}), ({6}, {5 }) }.

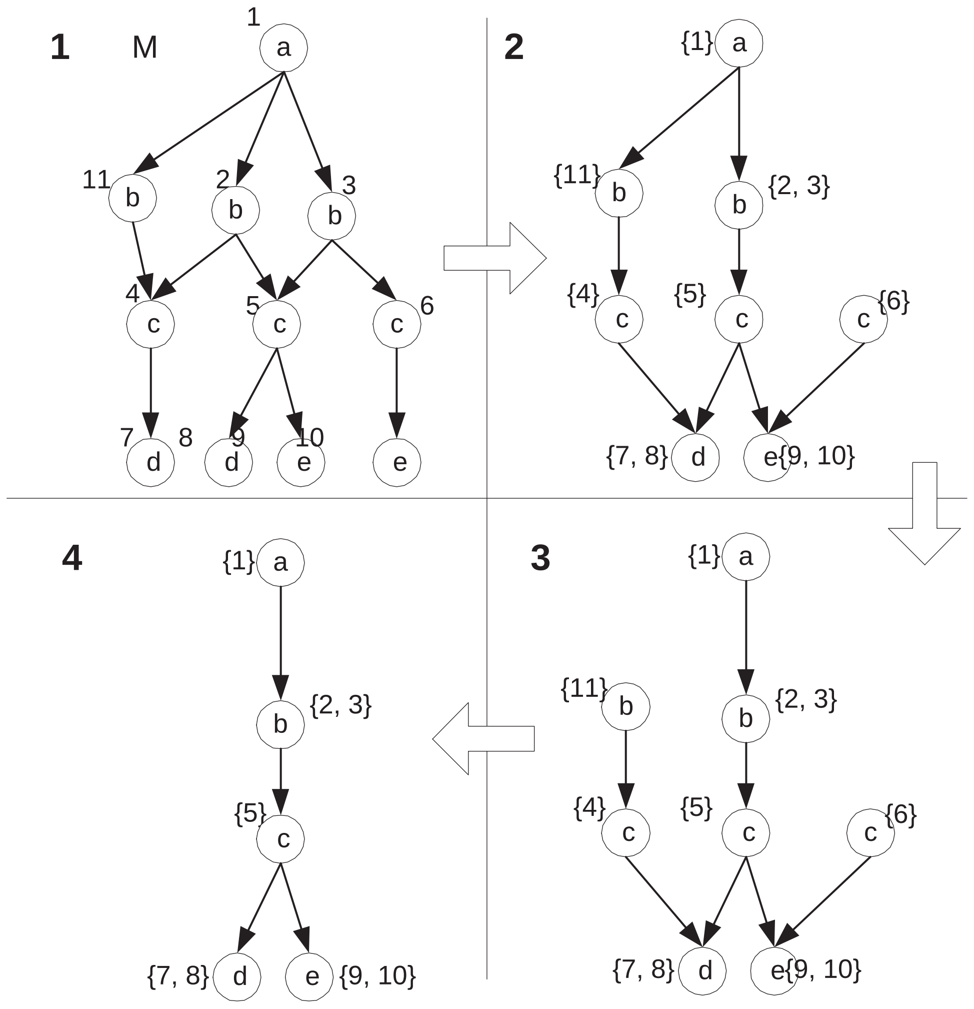


Figure 5.5: An example of the Minimization Algorithm

The next step of the algorithm disconnects all little brothers. $\{11\}$ is a little brother of $\{2, 3\}$ with $\{1\}$ as their father. Hence we remove the edge $(\{1\}, \{11\})$. The next and last step, outputs a reduced structure by removing all unreachable states such as $(\{11\}, \{4\}, \text{and } \{6\})$

5.2 The Quotient Technique

In this section we discuss the quotient method for finite-state systems in the specific case of parallel synchronization composition ($\|$) and simulation preorder satisfaction formalism ($\preceq$). For each quotient step, we adapt the equation solving method, proposed by Larsen and Xinxin [47], in order to find the largest quotient, with respect to $(\preceq)$.

5.2.1 The Model, Satisfaction Formalism, Parallel Composition

Definition 20 *A finite-state system is a tuple $FS = (S, s_0, \rightarrow)$, where S is a finite set of states, s_0 is the initial state and $\rightarrow \subseteq S \times Act \times S$ is a transition relation.*

Example 1 *Consider the deterministic finite-state A and B in Figure 5.6. The common set of actions is $Act = \{a, b, c, d\}$. Normally, a finite-state (i.e the finite-state A) is presented as a term over actions. Example: $a.(c.NIL+d.NIL)+b.Nil$ is the term corresponding to the left finite-state system of Figure 5.6. Often we shall omit trailing occurrences of NIL and simply write $a.(c+d)+b$.*

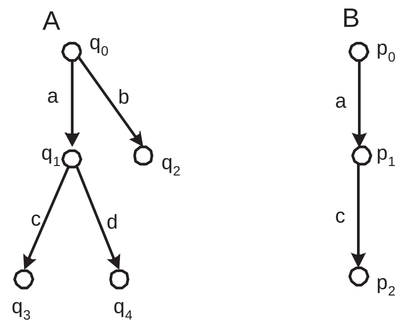


Figure 5.6: Two finite-state systems

Definition 21 *A relation $R \subseteq S \times S$ is a simulation preorder if for all $a \in Act$, the following holds: whenever $(p, q) \in R$*

- *if $p \xrightarrow{a} p'$ then $q \xrightarrow{a} q'$ for some q' s.t $(p', q') \in R$*

we write q simulates p, $p \preceq q$ if $(p, q) \in R$ for some simulation preorder R

Obviously, there is a simulation relation R in Example 1, where
$R = \{(p_0, q_0), (p_1, q_1), (p_2, q_3)\}$.

Definition 22 *Given two finite-states $S_1 = (S_1, {s_0}^1, \rightarrow_1)$, $S_2 = (S_2, {s_0}^2, \rightarrow_2)$. Then the parallel composition is a finite-state system $S = (S, s_0, \rightarrow)$, where $s_1 \| s_2 \in S$ whenever $s_1 \in S_1$ and $s_2 \in S_2$, $s_0 = {s_0}^1 \| {s_0}^2$, and the $\rightarrow$ is given by the following rule:*

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1' \qquad s_2 \xrightarrow{\alpha}_2 s_2'}{s_1 \| s_2 \xrightarrow{\alpha} s_1' \| s_2'}$$

Figure 5.7 shows the parallel composition of the two finite-state systems A and B in Example 1.

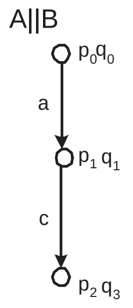


Figure 5.7: The synch parallel composition

5.2.2 The Quotient Structure

Given two finite-state systems A and C and a "finite-state" specification B we aim at constructing a specification $B//A$, called the quotient such that

$$A \| C \preceq B \text{ if and only if } C \preceq B//A$$

The bi-implication indicates that we are factoring parts of the parallel system into the specification. The quotient construction is defined as follows:

Definition 23 *Let $\mathbb{A}$ and $\mathbb{B}$ be finite-state systems, where $\mathbb{A} = (S_1, {s_0}^1, \rightarrow_1)$ and $\mathbb{B} = (S_2, {s_0}^2, \rightarrow_2)$. The quotient $\mathbb{B}//\mathbb{A}$ is the finite-state system $(S_1 \times S_2 \cup \{\top\}, s_0, \rightarrow)$, where $\forall \alpha \in Act. \top \xrightarrow{\alpha} \top$, such that $\rightarrow$ satisfies the following rules:*

- *if $A \overset{\alpha}{\nrightarrow}_1$ then $B//A \xrightarrow{\alpha} \top$.*
- *if $A \xrightarrow{\alpha}_1$ and $B \xrightarrow{\alpha}_2$ then $B//A \xrightarrow{\alpha} B'//A'$.*

The next theorem proves that (//) is indeed a quotient.

Theorem 6 *Whenever $A \in \mathbb{A}$, $B \in \mathbb{B}$ and for all finite-state X the following holds:*

$$A||X \preceq B \longleftrightarrow X \preceq B//A$$

Proof: We first prove the implication ($\longleftarrow$):

$$A||X \preceq B \longleftarrow X \preceq B//A \ (1)$$

Let $R = \{(A\|X, B) \mid X \preceq B//A\}$
Clearly, "$\longleftarrow$" is solved if only if R is a simulation relation.
For all $(A\|X, B) \in R$, assume $A\|X \overset{\alpha}{\rightarrow} Y$

then $A \overset{\alpha}{\rightarrow} A'$ and $X \overset{\alpha}{\rightarrow} X'$ with $Y = A'\|X'$.

But $X \preceq B//A$ and $X \overset{\alpha}{\rightarrow} X'$

then $B \overset{\alpha}{\rightarrow} B'$ with $B//A \overset{\alpha}{\rightarrow} B'//A'$ hence $X' \preceq B'//A'$.

Therefore, for all $(A\|X, B) \in R$, we have:

$A\|X \overset{\alpha}{\rightarrow} A'\|X'$, $B \overset{\alpha}{\rightarrow} B'$ then $(A'\|X', B') \in R$ with $X' \preceq B'//A'$.

Secondly proving the implication ($\longrightarrow$):

$$A||X \preceq B \longrightarrow X \preceq B//A \ (2)$$

Let $R = \{(X, B//A) \mid A\|X \preceq B\}$. Clearly, "$\longrightarrow$" is solved if only if R is a simulation relation.

For all $(X, B//A) \in R$, assume $B//A \overset{\alpha}{\rightarrow} Y$, from Definition 23:

- If $Y = \top$, then $A \overset{\alpha}{\not\rightarrow}$, since $A||X \preceq B$ then $X \overset{\alpha}{\rightarrow} X'$. Therefore both $B//A \overset{\alpha}{\rightarrow} T$ and $X \overset{\alpha}{\rightarrow} X'$, definitely $(X', \top) \in R$ with $A'\|X' \preceq B'$.
- If $Y = B'//A'$, then $(A \overset{\alpha}{\rightarrow} A'$, $B \overset{\alpha}{\rightarrow} B')$,
  but $A\|X \preceq B$ then $X \overset{\alpha}{\rightarrow} X'$, then $A\|X \overset{\alpha}{\rightarrow} A'\|X'$ with $A'\|X' \preceq B'$.

Therefore, for all $(X, B//A) \in R$, we have:

$X \overset{\alpha}{\rightarrow} X'$, $B//A \overset{\alpha}{\rightarrow} B'//A'$ then $(X', B'//A') \in R$ with $A'\|X' \preceq B'$.

From (1) and (2), we have proved the Theorem 6.

✠

Figure 5.8 illustrates the quotient structure of the two finite-state systems A and B in Example 1.

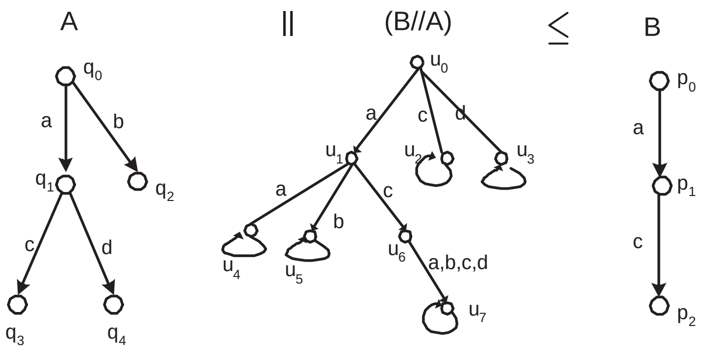


Figure 5.8: The synch parallel composition

5.2.3 General Quotient Structure Algorithm

In the previous section, we discussed the method for constructing the quotient structure in the case of parallel synchronization. However, in this section, we give an algorithm that computes the quotient structure for deterministic and acyclic systems, for all the cases of parallel composition, i.e. interleaving , synchronization and mixed synchronization. In this case, the idea of solving a set of equations ([47]) is extended to set of inequations with the two main problems being:

1. Solutions are not always guaranteed to exist, for instance:

$$a.Nil\rceil\lceil X \preceq b.Nil$$

2. It is necessary to consider *sets of* inequations rather than just single inequations. For instance:

$$a.Nil\rceil\lceil X \preceq a.b.Nil + b.a.Nil$$

implies that the solution must satisfy the following inequation as well

$$Nil\rceil\lceil X \preceq b.Nil$$

Figure 5.9 shows the main part of the algorithm, that generates a quotient structure X from the initial inequation $A\rceil\lceil X \preceq B$. The idea is that the algorithm starts with the initial set of inequation $E = \{A\rceil\lceil X \preceq B\}$. Thereafter, it iteratively checks all actions a, where $A \xrightarrow{a} A'$ and $B \xrightarrow{a} B'$, if $X \xrightarrow{a} X'$ such that $A\rceil\lceil X \preceq B$ by calling the function $TransOK(a, E)$. If there exists such an action a, the algorithm creates a new set of inequation by performing the function $Derive(a, E)$. A new set of inequation E' is derived from E by simultaneously transforming A, X and B with the same action a for all inequations in the set E. Finally, the algorithm returns a structure from the derived inequation, which is exactly the structure of the quotient X. Figure 5.10, 5.12, 5.11, 5.13 respectively show the pseudo-code of the functions $TransOK(a, E)$, $Derive(a, E)$, $Close(E)$ and $Consistent(E)$.

```
Function Solve(E: Inequation System): {N, $\cup$, $Undef$ : Node}

If Consistent ($E$) then

 Create New Node N;

     If E = $\emptyset$ then Return $N = \cup$;
     Else for ($\forall a \in \sum$ )
             If TransOK(a, E) then
              $E'$:= Derive (a, E);
              $E'$:= Close($E'$)
              $N'$ := Solve($E'$);
              If ($N \neq Undef$) then, Add $a$-edge $N \xrightarrow{a} N'$

Else Return $Undef$;

Endfunction
```

Figure 5.9: The main function for computing the Quotient

Function: TransOK (a, E)
Purpose: This boolean subfunction checks for each inequation: $A\rceil\lceil X \preceq B \in E$, if X is allowed to perform an action a such that $A\rceil\lceil X \preceq B$. The function returns *true* if all the inequations in E satisfy the above requirement.

```
Function TransOK $(a, E)$ : boolean

 $Flag$ := true;

    While $(\exists\ 'A\rceil\lceil X \preceq B' \in E)$ and $(Flag$= true$)$ do
    in case of:
        1. interleaving :
           • if B $\overset{a}{\nrightarrow}$ and $A \overset{a}{\nrightarrow} A'$ then $Flag := false$
        2. synchronization :
           • if B $\overset{a}{\nrightarrow}$ and $A \overset{a}{\rightarrow} A'$ then $Flag := false$
        3. Mixed-Synch :
           • for $\forall a \in Act_A / Act_B$
             if B $\overset{a}{\nrightarrow}$ then $Flag := false$
           • for $\forall a \in Act_A \cap Act_B$
             if B $\overset{a}{\nrightarrow}$ and $A \overset{a}{\rightarrow} A'$ then $Flag := false$
    Endwhile

Return $Flag$ ;

Endfunction
```

Figure 5.10: Function TransOK(a,E)

Function: Derive(a, E)
Purpose: This function returns an new inequation set. The new inequation is derived from E with action a.

```
Function Derive $(a, E)$
 $E' = \{\emptyset\}$;
for ( $\forall' A\rceil\lceil X \preceq B' \in E$ )
        If $(A\rceil\lceil aX' \preceq B \xrightarrow{a} A'\rceil\lceil X' \preceq B')$ then
        Add $(A'\rceil\lceil X' \preceq B', E')$;
Return $E'$ ;
Endfunction
```

Figure 5.11: Function Derive(a,E)

Function: Close(E)
Purpose: Add all inequations which can be derived, without involving X, from E to the same inequation set, E, i.e. if $(A\rceil\lceil X \preceq B) \in E$, such that $A \xrightarrow{a} A', B \xrightarrow{a} B'$ then add $(A'\rceil\lceil X \preceq B')$ to E.

```
Function Close(E)
        $flag$ := true;
      while $(\exists\, 'A\rceil\lceil X \preceq B' \in E)$ and ($flag$= true) do
        $flag$ := false;
            for ( $\forall a \in \sum$) do
                if $(\exists\, a.A'\rceil\lceil X \preceq a.B' \in E)$ then
                    Add($A'\rceil\lceil X \preceq B', E$)
                    $flag$ := true;
                endif
      endwhile
return $E$;
EndFunction
```

Figure 5.12: Function Close(E)

Function: Consistent(E)
Purpose: This function checks for each inequations $'A\rceil\lceil X \preceq B' \in E$, with all actions $a \in Act$, if $A\rceil\lceil X \xrightarrow{a} A'\rceil\lceil X'$ then $B \overset{a}{\to} B'$. The function returns *true* if all the inequations all satisfy the above condition.

```
Function Consistent (E): Boolean
    $flag$:= true;
While ($\exists\ 'A\rceil\lceil X \preceq B' \in E$) and ($flag$= true) do
    for ( $\forall a \in Act$ )
    If $A\rceil\lceil X \xrightarrow{a} A'\rceil\lceil X'$ and ($B \overset{a}{\nrightarrow}$)then $flag$:= false;
Endwhile
Return $flag$ ;
```

Figure 5.13: Function Consistent(E)

We now consider an example as in the case of mixed synchronization (|||), see Figure 5.14.

Example: Let $\sum_1 = \{a, b\}$ be the set of actions in finite-state A, and $\sum_2 = \{b, c\}$ the set of actions allowed in finite-state X, with $\sum_3 = \{a, b, c\}$ for B.

The algorithm starts with an initial inequation in the set of inequation $E = \{a.b.Nil|||X \preceq a.b.c.Nil\}$ and performs the Close(E) function to close the inequations in E. The algorithm is recursive and runs on itself till the inequation set is empty or can no more be derived. Recall that X can only perform actions $\{b, c\}$. In the first investigation, action c is not allowed by the inequations in the set E_0, therefore E_0 can only be transformed by b to E_1. In the next investigation by action b, it is easy to see that E_2 is an empty set of inequations so that the algorithm returns the universal state at that stages. However, it continues checking E_1, which can perform both actions b and c.

Along the derivation of the next set of inequations, the algorithm systematically builds the quotient structure. Finally, the algorithm returns this structure as X, which is based on the derived inequation systems, see X in Figure 5.14.

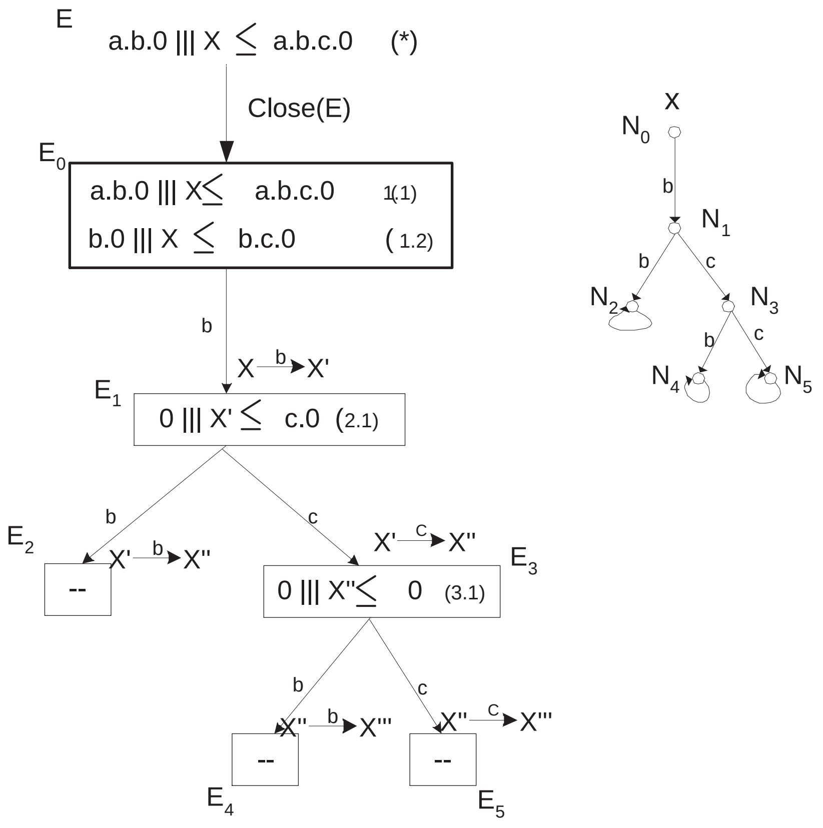


Figure 5.14: An example of Mixed Synchronous Parallel Composition

In Figure 5.15, we can check the quotient X, which is found in Figure 5.14, is the solution of $A|||X \preceq B$.

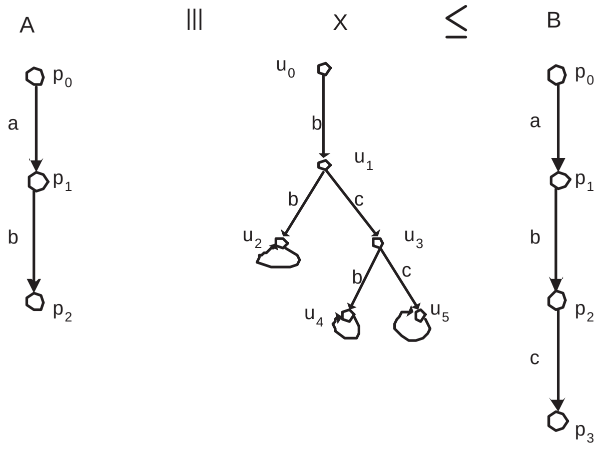


Figure 5.15: An example of Mixed Synchronous Parallel Composition

Chapter 6

Minimization for PTSs

In this chapter we adapt the Minimization Algorithm to the Probabilistic Transition System (PTS) and aim at generating a reduced PTS structure by this process. The input is a PTS and the output is a single PTS, with no equivalent classes and hopefully smaller than the input.

The minimization achieved here is intended to be used as heuristic in the Quotienting Technique to further reduce the size of the structure of the transformed specification.

6.1 Probabilistic Transition Systems

From Definition 4 which gives a concise definition of a PTS, we begin by first examining the differences between a PTS and a Kripke structure. We, however, use as input, a Rooted Probabilistic Transition Systems (RPTSs), which is a PTS with a specified initial state, s_0. Hence the tuple $(S, s_0, \rightarrow, V)$.

Given a Kripke structure $M = (S, s_0, R, L)$ and a rooted PTS $T = (S, s_0, \rightarrow, V)$, the differences are

- T has a finite set of states as M.
- the transition function of
    - M: $R \subseteq S \times S$,
    - T: $\rightarrow \subseteq S \times Act \times Dist(S)$, is a finite transition relation, where Act is a finite set of actions and $Dist(S)$ is a finite distribution over the states S.
- both have the same labelling function.

The main challenge in considering PTSs is their transition function, which is from states, by actions and onto distributions over states. Care must be taken in making transitions when minimizing the structure.

6.1.1 The Size of a PTS structure

Let us now consider what the size of a PTS should be. We consider size in terms of the number of states, actions, and distributions.

Theorem 7 *For every transition out of a state by a unique action, there is one and only one associated distribution. There are never more distributions than states in a given PTS.*

Definition 24 *Let $|T|$ denote the size of a PTS $T = (S, \rightarrow_T, V)$. Then $|T|$ is the pair $(|S|, |\rightarrow_T|)$ where $|S|$ is the number of states in T and $|\rightarrow_T|$ is the number of transition (induced distributions) in T.*

Definition 25 *Let $|T|$ be the size of a PTS T. We say that $|T| \leq |T'|$ if either of the following holds:*

1. $|S| \leq |S'|$ *or*
2. $|S| = |S'| \wedge |\rightarrow_T| \leq |\rightarrow'_T|$
   where $|\rightarrow_T|$ is the number of transitions in T, from each state.

As an example, the PTSs in Figure 6.1 below illustrate the first case where the size of the PTS is determined by the number of states. Hence $|T| \leq |T'|$ because $|S| \leq |S'|$.

In the next example, the PTSs in Figure 6.2 illustrate the second case where the size of the PTS is determined by the number of distributions. Hence $|T'| \leq |T''|$ because $|S| = |S'|$ and $|\rightarrow_T| \leq |\rightarrow'_T|$.

We now define what simulation equivalence means for a PTS structure and also for a state in a PTS.

Definition 26 *A simulation between two rooted PTSs $T = (S, \rightarrow, V, s_0)$ and $T^\star = (S^\star, \rightarrow^\star, V^\star, s_0^\star)$ is a subset $\mathcal{R}$ of $S \times S^\star$ such that*

1. $(s_0, s_0^\star) \in \mathcal{R}$

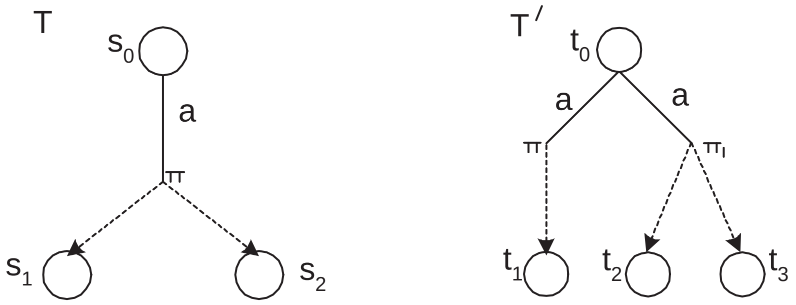


Figure 6.1: The Size of a PTS by number states

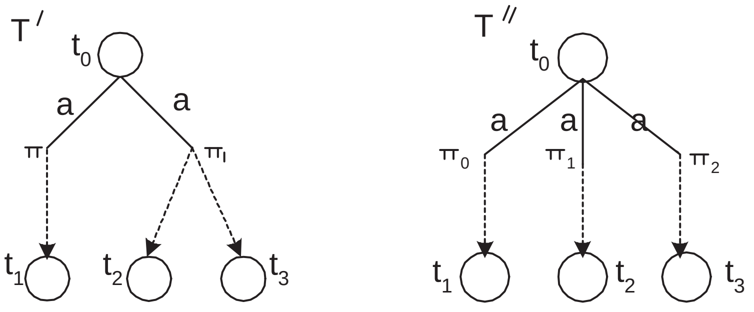


Figure 6.2: The Size of a PTS by number of distributions

2. whenever $(s, s^\star) \in \mathcal{R}$ and $s \overset{\alpha}{\rightarrow} \mu$ then there exists a transition $s^\star \overset{\alpha}{\rightarrow}^\star \mu^\star$ and a weight function δ for $(\mu, \mu^\star)$ with respect to $\mathcal{R}$ i.e. $\mu \sqsubseteq_{\mathcal{R}} \mu^\star$.

We say s is simulated by $s^\star$(denoted by $s \sqsubseteq s^\star$) iff there exists a simulation that contains $(s, s^\star)$. We say that s and $s^\star$ are simulation equivalent (denoted by $s \equiv s^\star$)if ($s \sqsubseteq s^\star$) and ($s^\star \sqsubseteq s$). Consequently, $T \sqsubseteq T^\star$ if the above conditions hold, and $T \equiv T^\star$ if the initial states are equivalent, i.e. $s_0 \equiv s_0^\star$.

In Figure 6.3, the simulation relation will include the pairs $(s_1, s_7), (s_7, s_1)$ and the trivial pairs. We can then conclude that $s_1 \equiv s_7$.

In constructing an equivalent class, all equivalent states belong to the same class, such that it is the set of states defined by the equivalence relation on them. The class that contains the initial state, s_0, becomes the initial equivalent class of the PTS. If there are no equivalent states, each state is then a unique class and every equivalent class contains a single state.

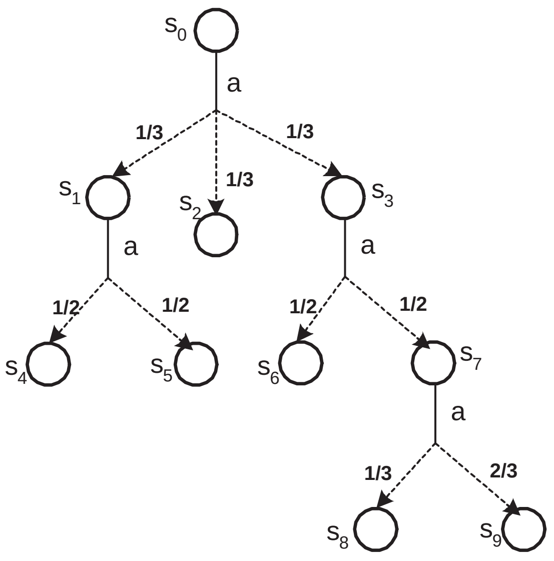


Figure 6.3: Simulation equivalence in a PTS

We now examine, the issue of a "little brother" in the case of a PTS, over states and distributions, which we denote by $\llcorner$. Here too, we assume a priority, first on distributions and then by state.

Definition 27 *1. Distribution-wise:* $\pi_1 \llcorner \pi_2$ *if* $\exists s_3$, $\exists a$, *such that* $s_3 \xrightarrow{a} \pi_1, s_3 \xrightarrow{a} \pi_2$, *with* $\pi_1 \sqsubseteq_R \pi_2; \pi_2 \not\sqsubseteq_R \pi_1$.

2. State-wise: $s_1 \llcorner s_2$ *if* $\exists s_3$, $\exists \pi$, $\exists a$, *such that* $s_3 \xrightarrow{a}_p \pi(s_1), s_3 \xrightarrow{a}_q \pi(s_2)$, *with* $s_1 \sqsubseteq s_2; s_2 \not\sqsubseteq s_1$.

We illustrate this with an example, where we disconnect an edge, between states because a distribution (Figure 6.4) or a state (Figure 6.5) is a *little brother* of another.

From the "*distribution*"-little brother elimination of Figure 6.4, it can be seen that $T \equiv T^\star$ (because $T \sqsubseteq T^\star$ and $T^\star \sqsubseteq T$) whereas for the "*state*"-little brother elimination of Figure 6.5 only $T \sqsubseteq T^\star$ (and in general not $T^\star \sqsubseteq T$). Theorem 8 clearly states this observation. Disconnecting little brothers by distribution, guarantees this equivalence, although it might not be the smallest reduced structure. However disconnecting by states does not guarantee this equivalence. In general, $T \sqsubseteq T^\pm$ but $T^\pm \not\sqsubseteq T$. In the example in Figure 6.5, this can be observed in the pairs $(t_1, {t_2}^\pm), (t_2, {t_2}^\pm) \in R$ and $({t_2}^\pm, t_2) \in R$ but not $({t_2}^\pm, t_1)$.

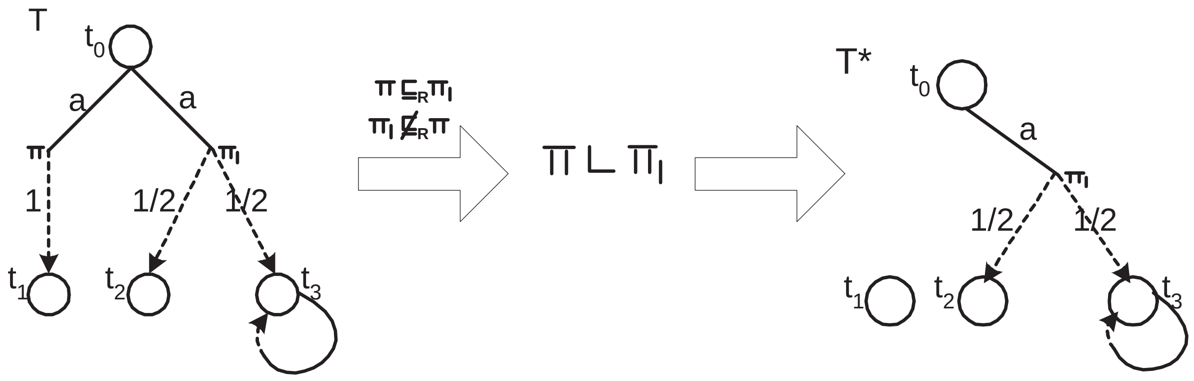


Figure 6.4: Disconnecting Little Brother by distribution

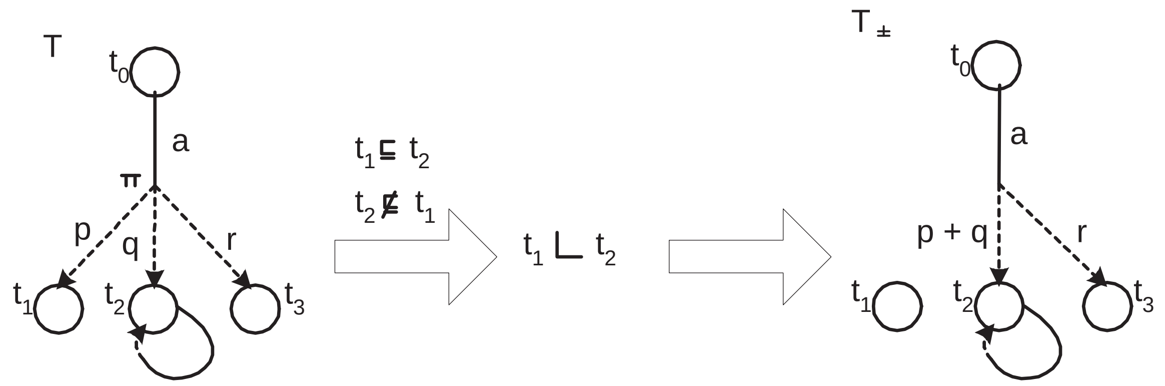


Figure 6.5: Disconnecting Little Brother by state

Definition 28 *A PTS T is reduced if:*

1. *There are no simulation equivalent states in T*
2. *There are no distributions π_1, π_2 such that π_1 is a "little brother" of π_2. This is the main priority. "Little brother" of states may be allowed.*
3. *All states in T are reachable from the initial state(class).*

Theorem 8 *For any PTS T there exists a reduced PTS T^* such that $T \sqsubseteq T^*$ and $T^* \sqsubseteq T$ by removing all "little-brother distributions".*

6.2 The Probabilistic Minimizing Algorithm

There are three steps as in the original algorithm. Consideration must be given to:

1. Building the $\forall-$ quotient, by constructing the simulation classes.
2. Identifying little brothers in terms of distributions and states, and disconnecting them.
3. Removing unreachable states in T.

6.3 Generating a Reduced PTS

STEP 1: Building the $\forall-$ Quotient.

Building the $\forall-$ quotient, by constructing the probabilistic simulation classes.

Constructing the Equivalent Classes
: build the maximal simulation relation, $H_M : S \times S'$ over T to find the equivalent states. If $(s_1, s_2) \in H_M$ and $(s_2, s_1) \in H_M$ then s_1, s_2 are simulation equivalent, denoted, $s_1 \equiv s_2$. Hence s_1, s_2 will belong to the same class. If the set of equivalent states is null and empty, that is, there are no equivalent states, then each class contains a single state.

Definition 29 *The $\forall-$ quotient PTS, $T_q =< S_q, \longrightarrow_q, V_q, s_0q >$ is defined as*

- *S_q is the set of equivalent classes of the simulation equivalence*
- $\longrightarrow_q = \{(\alpha_1, \Pi) | \forall s \in \alpha_1 . \exists \pi, s \xrightarrow{a} \pi . \forall \alpha_2, \Pi(\alpha_2) = \sum_{s' \in \alpha_2} \pi(s')\}$
- $V_q([s]) = V(s)$
- $s_0q = [s_0q]$

At the end of this step, the output structure must have no equivalent states, since this PTS, contains the distinct and independent equivalent classes, which are the states of the PTS. Figure 6.6

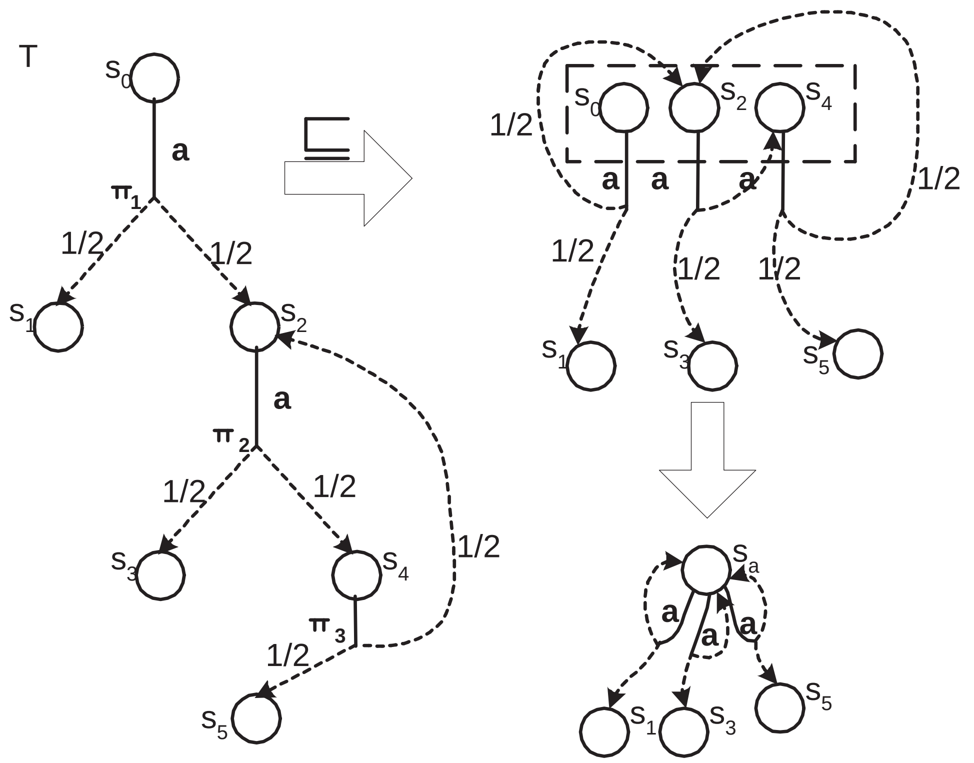


Figure 6.6: Constructing the $\forall-$Quotient PTS

STEP 2: Disconnecting Little Brothers

We need to clarify the difference between a transition and a distribution.

A transition is from a state by an action over a distribution to some states. Hence in our diagrams, this is denoted by the lines from the state plus the dotted lines to states. Distributions are denoted by the dotted lines.

(a) Distribution-wise (disconnect the transition)
If $\exists s$, $\exists a$, such that $s \xrightarrow{a} \pi_1, s \xrightarrow{a} \pi_2$, with $\pi_1 \sqsubseteq \pi_2; \pi_2 \not\sqsubseteq \pi_1$, such that $\pi_1 \llcorner \pi_2$, then remove the transition $s \xrightarrow{a} \pi_1$.

(b) State-wise (disconnect distribution and add up probability to *big brother*)
If $\exists s_3$, with $s_3 \xrightarrow{a} \pi_p(s_1), s_3 \xrightarrow{a} \pi_q(s_2)$, and $s_1 \sqsubseteq s_2; s_2 \not\sqsubseteq s_1$, such that $s_1 \llcorner s_2$, then remove $\pi(s_1)$ and put $\pi(s_2)$ as $\pi_x(s_2)$ where $x = p + q$.

From our example, we eliminate little brother distributions from Figure 6.7(a).

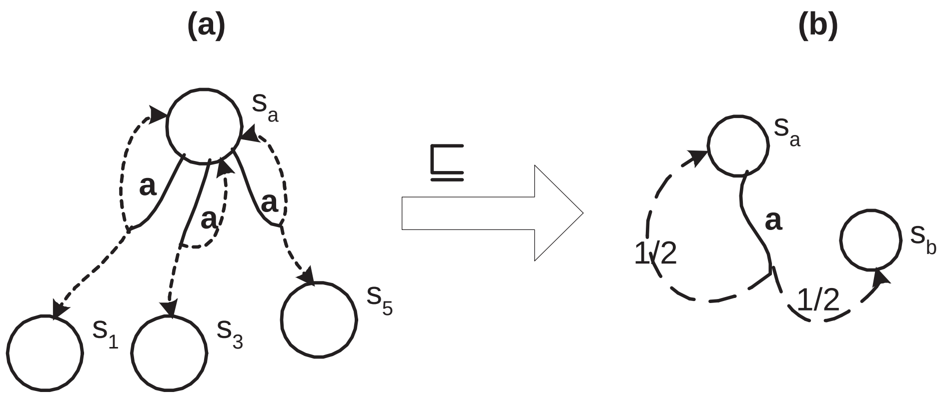


Figure 6.7: Disconnecting Little Brothers in a PTS

STEP 3

Remove all unreachable states, these are the states that are unreachable from $[s_0]$.

The resulting PTS from these three steps is a reduced PTS. See Figure 6.8

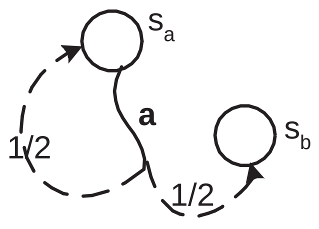


Figure 6.8: The Reduced PTS

And indeed this reduced structure simulates the original in Figure 6.6.

Chapter 7

The Quotient Technique for PTS

The quotient technique has been successfully applied for finite-state [3] and real-time state systems [35]. In this chapter we shall investigate the quotient technique for the probabilistic labelled transition system, where the specification formalism is a specific blocking PTS (bPTS). The idea of using the bPTS specification is that, we consider compositions of deterministic acyclic probabilistic models (T) and a model for specification such as $T_1 || ... || T_n \sqsupseteq B$, where the bPTS B is able to present $\mu - calculus$ (e.g. $< a >_{\geq 0.2} < b >_{\geq 0.5} tt$). We wish to verify whether the parallel composition is at *least* the blocking PTS without having to construct the complete parallel system.

7.1 The Model

In this section, we define a specific model of PTS called the blocking Probabilistic Transition System(bPTS), which we use as a model for specifying our specifications. A state s is identified as a blocking state if it either is a sink state or an universal state. The definition of bPTS is consequently defined as follows:

Definition 30 *A PTS $T = (S \cup \{NIL\}, \rightarrow, V)$ is a blocking PTS if for all $s \in S$, $\alpha \in Act$ there exists at most one transition $s \xrightarrow{\alpha} \pi$ where π is a blocking distribution over a pair of states $s' \in S$ and NIL in the sense that $\pi(s') + \pi(NIL) = 1$. And for all $a \in Act, NIL \overset{a}{\not\rightarrow}$.*

Figure 7.1 illustrates a blocking PTS.

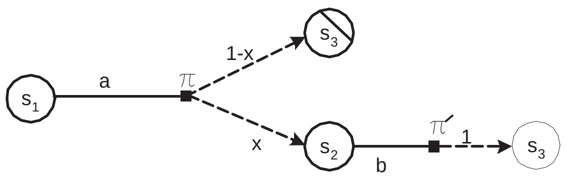


Figure 7.1: An example of blocking PTS

7.2 The Logic and its bPTS Representation

The blocking PTS model is defined to present a fragment of PCTL such as "after a request for a number of tasks, there is at least a 50 percent probability that the first task is done and after that with at least a 30 percent probability the second task is carried out". The bPTS in Figure 7.2 presents the following property example: $< a >_{\geq 0.2} < b >_{\geq 0.5} < c >_{\geq 1} tt$

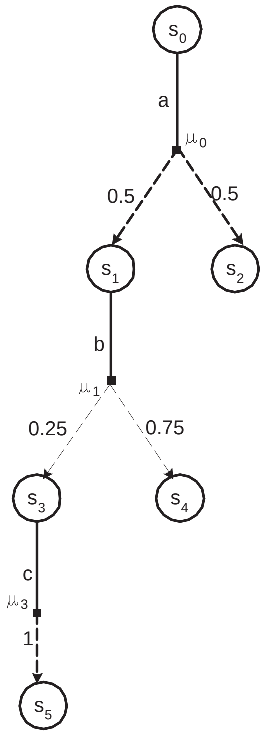


Figure 7.2: The bPTS presentation of a property

7.3 Quotient Structure

Before we consider the quotient structure, let us examine some implications of our specification and its representation, the bPTS.

Given

$$A||X \sqsupseteq B \tag{7.1}$$

where A is deterministic, acyclic and B is blocking and we require X to be the smallest structure possible such that the equation holds.

First of all, we need to check if Equation 7.1 has any solutions at all. Is there an X, for which Equation 7.1 holds?

We begin by defining a *universal* state (U) which allows transitions by all actions $a \in Act$, from it by a distribution to the same *universal* state. That is:

$$U \xrightarrow{a} \pi_u \text{ where } \pi_u(U) = 1 \text{ and } \pi_u(s) = 0 \text{ whenever } s \neq U.$$

Let $\{U : \forall a \in Act, U \xrightarrow{a} U\}$. Then $U \sqsupseteq C$ for all PTS C.

It suffices to check whether $A \sqsupseteq B$, in order to establish that Equation 7.1 has solutions. Example, there does not exist an X, for which the equation

$$NIL||X \sqsupseteq a.NIL$$

holds true.

The $\sqcup$−Construct

Definition 31 *Let X_1 and X_2 be blocking PTSs. Then $X_1 \sqcup X_2$ is a blocking PTS given by*

$$X_1 \sqcup X_2 \xrightarrow{a} \begin{cases} \pi_1 & if X_1 \xrightarrow{a} \pi_1, X_2 \overset{a}{\not\rightarrow} \\ \pi_2 & if X_2 \xrightarrow{a} \pi_2, X_1 \overset{a}{\not\rightarrow} \\ \rho_{\pi_1\pi_2} & if X_1 \xrightarrow{a} \pi_1, X_2 \xrightarrow{a} \pi_2. \end{cases}$$

where $\rho_{\pi_1\pi_2}(X_1' \sqcup X_2') = max\{\pi_1(X_1'), \pi_2(X_2')\}$ and

$$\rho_{\pi_1\pi_2}(NIL) = 1 - max\{\pi_1(X_1'), \pi_2(X_2')\}.$$

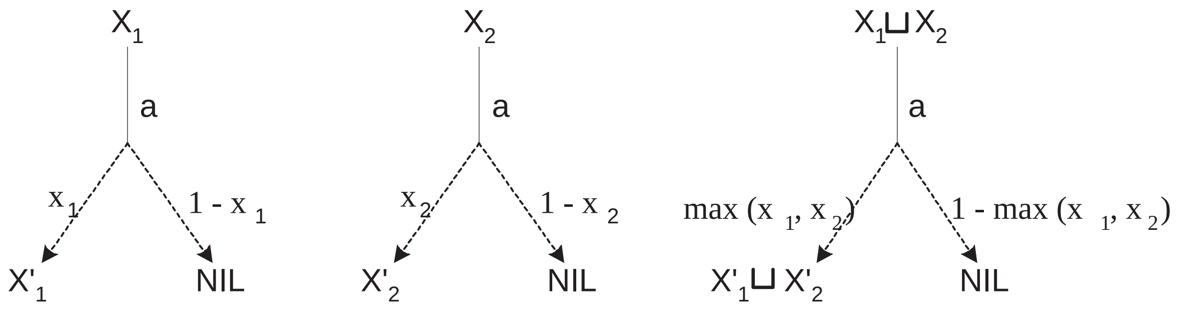


Figure 7.3: Properties of the bPTS

Lemma 1 *Whenever X_1 and X_2 are blocking PTSs, then $X_1 \sqcup X_2 \sqsupseteq X_1$ and $X_1 \sqcup X_2 \sqsupseteq X_2$.*

Moreover if for a blocking PTS Y, $Y \sqsupseteq X_1$ and $Y \sqsupseteq X_2$, then $Y \sqsupseteq X_1 \sqcup X_2$.

Figure 7.3 illustrates X_1, X_2 and $X_1 \sqcup X_2$.

Lemma 2 *If $A_1 || X_1 \sqsupseteq B$ and $A_2 || X_2 \sqsupseteq B$ then $A_1 || (X_1 \sqcup X_2) \sqsupseteq B$ and $A_2 || (X_1 \sqcup X_2) \sqsupseteq B$.*

7.4 Algorithm for Computing the Quotient

In constructing the Quotient structure such that the Equation 7.1 holds, we will have to consider first, the structure of the PTS, with regards to the states and transitions to these states, which we represent by X, and secondly the probabilities of the distributions over the states, represented by x. We describe an informal approach to obtaining the quotient structure before giving an algorithm to obtain it, formally.

7.4.1 An Informal Approach

Let us consider Equation 7.1, $A || X \sqsupseteq B$, where A and B are given PTSs. Now consider an $a-$ transition of B. Obviously, in order for Equation 7.1 to have solutions, A should also have a unique $a-$transition. Below we display the $a-$derivative in Figure 7.4

For each i check whether

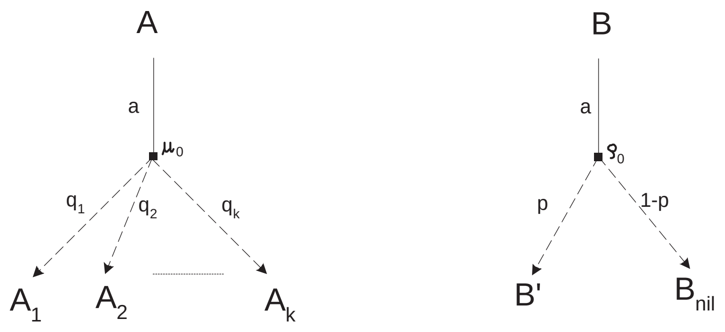


Figure 7.4: A unique a-derivative

$$A_i \sqsupseteq B'$$

Let $I = \{i | A_i \sqsupseteq B'\}$.
For each $i = I$, construct X'_i such that $A_i || X_i \sqsupseteq B'$
Let

$$X' = \bigsqcup_{i \in I} X'_i$$

because then $A_i || X' \sqsupseteq B'$ for all $i \in I$,

$$\implies A_i || X' \sqsupseteq A_i || X_i \sqsupseteq B'.$$

The probability of the Distribution (x)

The goal is to find the (smallest) probability x for the structure X satisfying $A_i || X' \sqsupseteq B'$ for all $i \in I$. The problem is reduced to that of finding flow in the induced network, such that Equation 7.1 holds. Figure 7.5 illustrates the network.

To find the minimal x such that the maximum flow in the network of Figure 7.5 is 1,

$$p \leq x \cdot \sum_{i \in I} q_i$$

$\implies$

$$\frac{p}{\sum\limits_{i \in I} q_i} \leq x$$

And since $0 \leq x \leq 1$, the minimum value for x is

$$x = \frac{p}{\sum\limits_{i \in I} q_i}$$

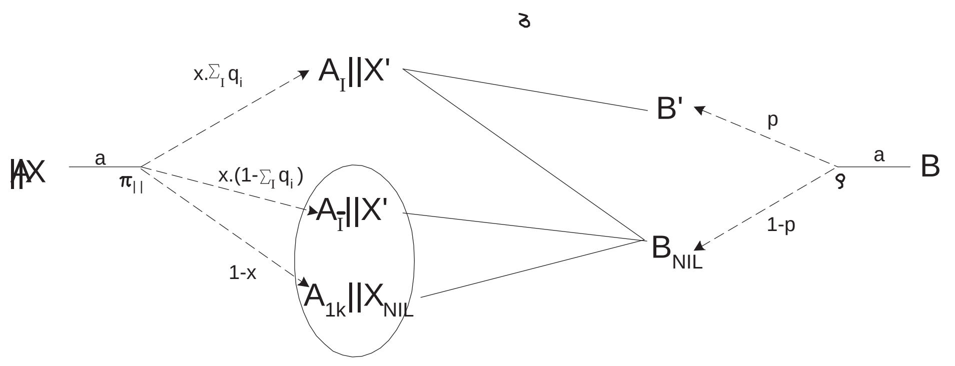


Figure 7.5: Preservation of weight function

7.4.2 A Formal Approach

Definition 32 *Let A be a deterministic and acyclic PTS and let B be a blocking PTS then*

$$Solvable(A, B) = \begin{cases} true; & if A \sqsupseteq B \\ false; & otherwise \end{cases}$$

Lemma 3 *Let A be a deterministic and acyclic PTS and let B be a blocking PTS then $Solvable(A, B) = true$ iff for some blocking PTS C, it holds that $A||C \sqsupseteq B$.*

Proof

$\Longrightarrow$
Then $A \sqsupseteq B$ but $A||U \sqsupseteq A$ where $U \xrightarrow{a} - - \overset{1}{-} - > U$, for all $a \in Act$. Clearly $U \sqsupseteq C$ for all PTSs C and U is a blocking PTS.

$\Longleftarrow$
Then $A||C \sqsupseteq B$ for some PTS C. But then $A||U \sqsupseteq B$ (as $U \sqsupseteq C$) But $A||U = A$, so $A \sqsupseteq B$.

✠

Purpose: Given a deterministic, acyclic PTS A and a blocking PTS B, BUILD constructs a blocking PTS X, that will solve $A||X \sqsupseteq B$ or BUILD returns "no solution found". We present a pseudo-code for the algorithm in Figure 7.4.2 and then subsequently explain it according to the line numbers.

Line 1 verifies if a solution for the Equation 7.1 can be found. The algorithm constructs the derivatives of X from Line 3.

```
BUILD(A, B)
 1. If not $SOLVABLE(A, B)$
 2. then return "no solution found"
 3. Else
 4.     For ( $\forall a \in \sum$) do
 5.         If $B \overset{a}{\not\rightarrow}$ then $X \overset{a}{\not\rightarrow}$
 6.         Else
 7.             For each $(i \in I)$
 8.             $X_i' := BUILD(A_i, B_i)$
 9.             $X' := \sqcup_{i \in I} X_i'$
10.         Endif
11. Endif
12. Return $X$;
13. End
```

Figure 7.6: The BUILD Algorithm

In Line 6, $B \xrightarrow{a}, A \xrightarrow{a}$, otherwise $Solvable(A, B)$ would have returned $false$ in Line 1. We consider the unique $a-$derivatives of A and B as shown in Figure 7.4. We let $I = \{i | A_i \sqsupseteq B'\}$ and assume $I = \{1, \ldots, j\} \subseteq \{1, \ldots, k\}$.

X' in Line 9 is constructed as such, because for all i, $A_i || X' \sqsupseteq B'$ and Lemma 3.

In Line 9, the algorithm returns X as shown in the Figure 7.7

Theorem 9 *Let X be the result of $BUILD(A, B)$. Then $A || X \sqsupseteq B$.*

The proof is by induction on the depth of A.

Base: Let depth of $A = 0$, that is $A = NIL$.

But then as $SOLVABLE(A, B) = true$, $\Longrightarrow$ $B = NIL$ because $A \sqsupseteq B$.

Then obviously $X = NIL$, and

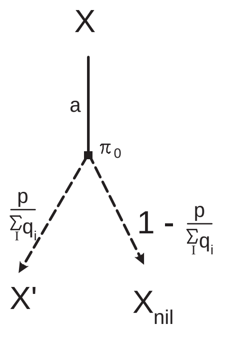


Figure 7.7: The Quotient Structure X

$NIL||NIL \sqsupseteq NIL.$

Induction Step

Let $B \xrightarrow{a} (p)(B') + (1-p)(NIL)$ as in Figure 7.4

By construction of X where $x = \frac{p}{\sum_{i \in I} q_i}$, as obtained previously and illustrated in Figure 7.5

By the induction hypothesis and Lemma 3, $A_i||X' \sqsupseteq B'$

The maximum flow between $A||X$ and B relative to action a can be initialized as in Figure 7.5

It is clear that the maximum flow is 1, hence the $a - transition$ of B is matched.

✠

Hence by applying the algorithm recursively, the quotient structure X for our equation, emerges.

Chapter 8

The Application of the Quotient Technique with Minimization

Having introduced separately the concepts of the Quotient Technique and Minimization, for PTSs, we now combine them, to see their application in the verification process.

8.1 Quotient and then Minimize

As already seen, the transformed specification also has the tendency to increase quickly in size. In this chapter, we explore using the minimization algorithm for PTSs as a simplification heuristic for this specification. From the basic equation

$A_1||A_2||A_3||\cdots||A_n \sqsupseteq B$ using the Quotient Technique,

$A_1||A_2||A_3||\cdots||A_{n-1} \sqsupseteq (B//A_n)^m$ where m is a minimization operator

Let $(B//A_n) = X_n$ and let $(X_n)^m = \mathbb{X}_n$, then

$A_1||A_2||A_3||\cdots||A_{n-1} \sqsupseteq (\mathbb{X}_n)$

$A_1||A_2||A_3||\cdots||A_{n-2} \sqsupseteq (\mathbb{X}_n//A_{n-1})^m$

$A_1||A_2||A_3||\cdots||A_{n-2} \sqsupseteq (\mathbb{X}_{n-1})$ where $(\mathbb{X}_n//A_{n-1})^m = (\mathbb{X}_{n-1})$

$\vdots$

The quotient structure $(B//A_n)$ should be noted as a single deterministic blocking PTS X_n.

In applying the algorithm to minimize this structure, we need to identify the equivalent classes in X_n.

Let us consider a fragment of a blocking PTS. Given the two blocking PTSs in Figure 8.1, we will like to consider simulation equivalence between the two states s_0 and t_0. Can we establish that $s_0 \sqsubseteq t_0$ and $t_0 \sqsubseteq s_0$, thereby conclude that $s_0 \equiv t_0$.

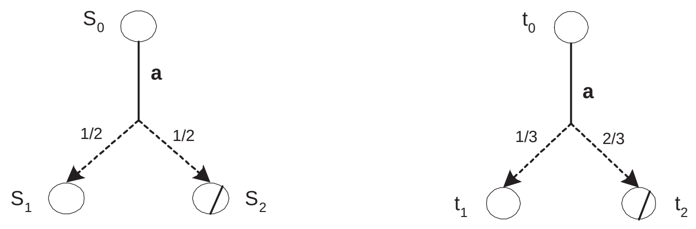


Figure 8.1: Simulation of two Blocking PTS

Using maximum flow in the network established, Figure 8.2, $s_0 \sqsubseteq t_0$ iff $p \leq q$. Also $t_0 \sqsubseteq s_0$ iff $q \leq p$. In this case, the latter is true. Obviously though, is the fact that both statements can not be true at the same time, unless $p = q$.

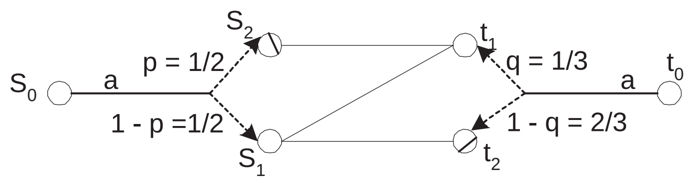


Figure 8.2: Flow Simulation of two Blocking PTS

This unassuming observation has very extensive implications. The first being that, we cannot obtain equivalence classes in the resulting blocking PTS of our quotienting. Hence every state is, and remains in a class of its own, which are all independent, without any intersections between them. Recall this is the first important step in the minimization algorithm.

The next step of the minimization step, involves the disconnection of distributions and states by the "little brother" phenomenon. Let us take a look at a typical X, Figure 8.3. Due to its deterministic nature, and the restrictive nature of our reachability properties, elimination of distributions by the little brother scenario, does not arise. In some cases, elimination of states can be possible, but as earlier stated, it is only by, distributions, that we guarantee an equivalence.

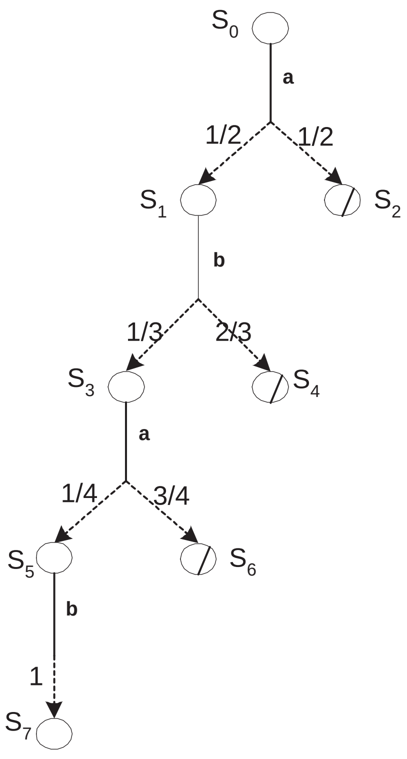


Figure 8.3: An X: Deterministic Blocking PTS

Although these observations are not very encouraging, the valid and most important conclusion we can draw is that, the resulting quotient structure obtained is indeed the smallest blocking PTS possible, with respect to simulation equivalence.

Theorem 10 *For a deterministic acyclic PTS A and a NIL-blocking PTS B such that $A||X \sqsupseteq B$, the quotient structure $X \sqsupseteq B//A$ is the ($\sqsupseteq$)-smallest component C, such that $A||C \sqsupseteq B$.*

The transformed specification is kept minimal and hence quotienting with bPTSs avoids the state explosion of the transformed specification, usually involved with the Quotienting Technique. Hence, with respect to our specification to be verified, represented with the blocking PTS, we do not need a simplification heuristic for the resulting transformed specification by the Quotienting Technique.

8.2 Minimize and then Quotient

However, we can still reduce the state space explored by looking at the model under consideration. This is our proposal. Given

$$A_1||A_2||A_3||\cdots||A_n \sqsupseteq B \tag{8.1}$$

Apply the Minimization Algorithm loosely to the individual components (or to clusters of components) in the model *before* starting with the Quotient Technique. The equation now becomes

$$(A_1)^m||(A_2)^m||(A_3)^m||\cdots||(A_n)^m \sqsupseteq B$$

$$\mathbb{A}_1||\mathbb{A}_2||\mathbb{A}_3||\cdots||\mathbb{A}_n \sqsupseteq B \text{ where } \mathbb{A}_n = (A_n)^m$$

By this the state space is reduced, even before quotienting. The transformed specification is thereby kept even smaller.

Let us explore this proposal with an example. Let the PTS in Figure 8.4 be an individual process (or the result of the parallel composition of some processes) in the model checking equation 8.1.

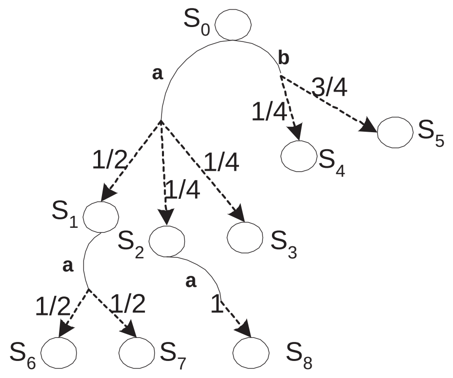


Figure 8.4: A deterministic acyclic PTS

Some minimization can be achieved in such models, by identifying some equivalent states. The states s_1, s_2 are equivalent and belong to one class. Also, the states s_6, s_7, s_8 will consequently belong to one class. Just at the first step of the algorithm, we can generate a $\forall$-structure as Figure 8.5

By considering this proposal, it is feasible to have space reductions by minimizing the components and plugging the reduced structures in place in the equation.

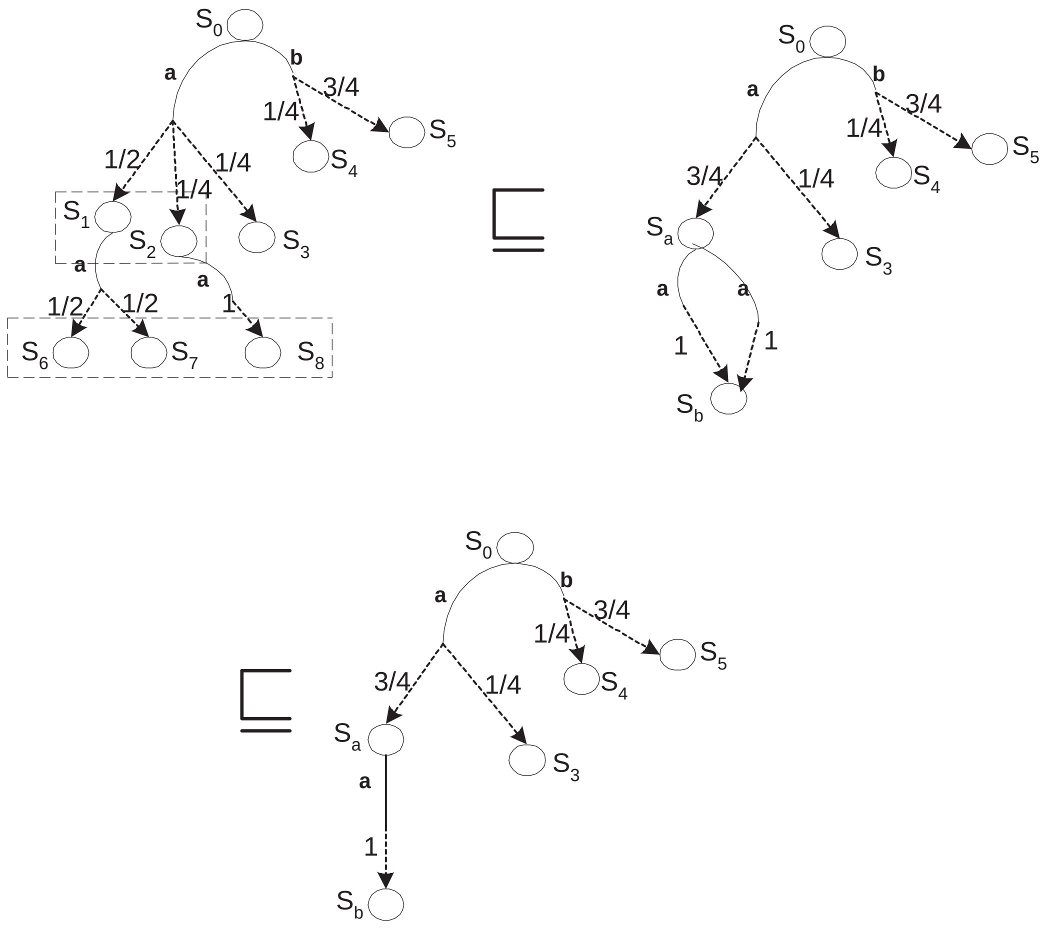


Figure 8.5: The ∀-structure PTS

Chapter 9

Implementation

The implementation is carried out in Visual C++, running in the Microsoft Windows environment. Some of the reasons for choosing Visual C++ on this project are its efficient libraries, tendency to increase productivity and easy to use design tools. Also one of our interest was to use the wizards for generating a fast user-friendly graphical application. We have decided to give our implementation a nice name CAPS (**C**ompositionality and **A**bstraction by **P**robabilistic **S**imlation) There are two main functionalities in CAPS.

1. Checking a right abstract(hopefully smaller) against a single PTS with respect to ($\sqsubseteq$). The considered algorithm involves application of the well-known Maximum Flow Problem. This functionality is called Compositional and Abstraction Checking.

2. Checking a right abstract in specific case of blocking PTS against a parallel deterministic and acyclic PTSs system, with respect to ($\sqsupseteq$), without having to construct the complete state space of the system by using the promising Quotient technique. This functionality is called Quotient Checking.

In the implementation, we have created crucial data structures such as PTSs and relation simulation structures, which are discussed in the next section. In Section 9.2 we examine the main modules of CAPS. In the rest of this chapter we give an instruction of a user's guide.

9.1 Data Structure

A complex PTS may need a huge space of memory, hence an efficient data structure for its storage is required. There are some compact structures such as in

RAPTURE [48] and *PRISM* [49], that are successful for presenting a PTS. However, we propose an acceptable data structure that defines our problem. The structure is set up to be able to store a number of states, a number of actions and a matrix whose elements hold a number of out going transitions $s \xrightarrow{\alpha} \mu$ and links to a substructure, where $Distr(s, \alpha)$ is established, see Figure 9.1. In addition, the simulation $R \subseteq S \times S'$ is presented by an adjacency matrix relation[S][S'], that is,

$$relationR[s][s'] = \begin{cases} 1 & \text{if } s \sqsubseteq_R s' \\ 0 & \text{otherwise} \end{cases}$$

whether $s \in S$ and $s' \in S'$.

```
struct PTSs {

        int nstates;

        int nactions;

        Transition trans[State][Action];

}; PTSs pts;

struct Transition {

        int nOut;

        float Distribution[TransitionOut][State];

};
```

Figure 9.1: Data Structure of a PTS

For instance, we use a variable *pts* with respect to the above data structure to present a PTS T in Figure 9.2. The number of states and the number of actions respectively are stored in pts.nstates = 7 and pts.actions = 3. Obviously, in order to present the number of out-going transitions from state s_0 with action α, we have pts.trans[s_0][α].nOut = 2, and to reach the transition $s_0 \xrightarrow{\alpha} \mu$, we can ask for pts.trans[s_0][α].Distribution[μ][S]= (0.25, 0.25, 0.5), where S is the set of states of the PTS T.

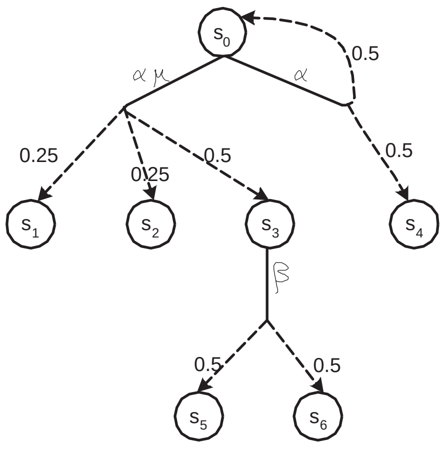


Figure 9.2: An example of a PTS

9.2 Modules Description

9.2.1 Maximum Flow Problem, Ford-Fulkerson Algorithm

As discussed the Ford-Fulkerson algorithm in Section 4.2, the complexity is O(n+m)nm, where n is the number of nodes and m is the number of edges in the graph G(N,E). In general, m = n^2 and the complexity is O(n^4+n^5). However, when the probabilistic simulation problem is reduced to the maximum flow problem, a network $\mathcal{N} = ((N, E), \perp, \top, c)$ is obtained, where G = (N,E) is actually a bipartite graph, written as $G = (X \cup Y, E)$. Therefore, to improve the complexity we implement an improved algorithm of Ford-Fulkerson for bipartite graph. The complexity of the improved algorithm is O(p^5), where p = max $\{\mid X \mid, \mid Y \mid\}$.

MaximumFlow($\mathcal{N}$)

Purpose:	The function computes the maximum flow value
Input:	A network $\mathcal{N} = ((X \cup Y), E, \perp, \top, c)$
Output:	The maximum value of flow in the network $\mathcal{N}$
Subroutine:	ResidualGraph($\mathcal{N}_{\mathcal{R}}$), AugmentingPath($\pi$)

This function performs the main task of the Ford-Fulkerson algorithm on the network $\mathcal{N}$. The function iteratively performs the two following steps: First, it generates a residual network $\mathcal{N}_{\mathcal{R}}$ by calling up the function ResidualGraph($\mathcal{N}_{\mathcal{R}}$), secondly it tries to find an augmenting path on graph $G = (X \cup Y)$ of $\mathcal{N}_{\mathcal{R}}$, where an augmenting path of a bipartite graph is the set of alternating nodes $x \in X$ and $y \in Y$. If there exists an augmenting path π the algorithm jumps to the function AugmentingPath(π), where the flow value is increased. The iteration ends when there is no augmenting path found.

ResidualGraph($\mathcal{N}_{\mathcal{R}}$)

Purpose:	Automatically generating a residual network of a given network $\mathcal{N}$
Input:	Network $\mathcal{N}$ with current flow $\mathcal{F}$
Output:	A residual network $\mathcal{N}_{\mathcal{R}}$

This function tries to establish a residual network $\mathcal{N}_{\mathcal{R}}$ of a given network $\mathcal{N}$ where the residual capacity of each edge $e \in E$ is assigned by $c(e)$ - $flow(e)$.

AugmentingPath(π)

Purpose:	The value of flow is increased by this function
Input:	The augmenting path π
Output:	Flow $\mathcal{F}$

In this function, the value of flow in $\mathcal{N}$, is increased by adding the current flow c_f to the flow along the path of the augmenting path π, where c_f is minimum residual capacity on the augmenting path π.

9.2.2 Computing Probabilistic Simulation

The algorithm for computing probabilistic simulation is shown in Section 4.1, its purpose is to establish the simulation relation set R of two given PTSs. The algorithm executes in polynomial time. The main functions are stated as follows:

SimulationPreorder(PTS, PTS')

Purpose:	This function computes the probabilistic simulation relation R of two given PTSs
Input:	Two PTSs $T = (S, \rightarrow, V)$, $T' = (S', \rightarrow', V')$
Output:	Probabilistic simulation $R \subseteq S \times S'$
Subroutine:	ConverttoMaxFlow(μ, μ'),

In this function, the main algorithm is implemented. All pairs $(s, s') \in S \times S'$ are examined to eventually return the simulation set R. In order to check for $s \sqsubseteq_R s'$, this function considers all transitions $s \xrightarrow{\alpha} \mu$, and searches in $Distr(s', \alpha)$ for a distribution μ', then checks whether $\mu \sqsubseteq_R \mu'$ by calling the function ConverttoMaxFlow(μ, μ'). If there is no corresponding μ' then (s, s') is removed from R by assigning R[s][s'] to false.

ConverttoMaxFlow(μ, μ')

Purpose:	This function converts probabilistic distributions μ and μ' to a network $\mathcal{N} = (N, E, \perp, \top, c)$ and answers the question of $\mu \sqsubseteq_R \mu'$.
Input:	Distributions μ, μ'
Output:	YES or NO
Subroutine:	MaximumFlow($\mathcal{N}$).

9.2.3 Compositional and Abstraction Checking

Let us now consider the function which applies the basic functions discussed above in order to check for a right abstraction.

isSimulated($R \subseteq S \times S'$)

Purpose:	This function check whether an abstract PTS T' simulates the original PTS Tor not
Input:	The simulation relation $R \subseteq S \times S'$
Output:	YES or NO

This function examine simulation R, if the pair of the two initial states $(s_0, {s_0}') \in R$ then $T \sqsubseteq T'$, otherwise $T \not\sqsubseteq T'$.

9.2.4 Quotient Checking

In order to construct the quotient of a blocking PTS B and a PTS T, we also use function $SimulationPreorder(T, T')$ in order to compute a simulation relation R. By doing that, and by generating the probability of the blocking PTS B, the function will return the construct of quotient $B//A$, which is a blocking PTS. In fact the structure of the quotient is the same as the structure of the specification blocking PTS B.

QuotientBuild(B: blockingPTS, A: PTS)

Purpose:	This function build the structure of the quotient $B//A$
Input:	The two PTS B and A
Output:	The quotient $B//A$
Subroutine:	$SimulationPreorder(T, T')$

9.3 Input File and Output Screen

9.3.1 Input File

An input file is a description of PTSs which are to be checked in CAPS. The format of an input file is shown in Figure 9.3 as an example. Particularly, the values of the distribution matrix present number of distributions that a state s

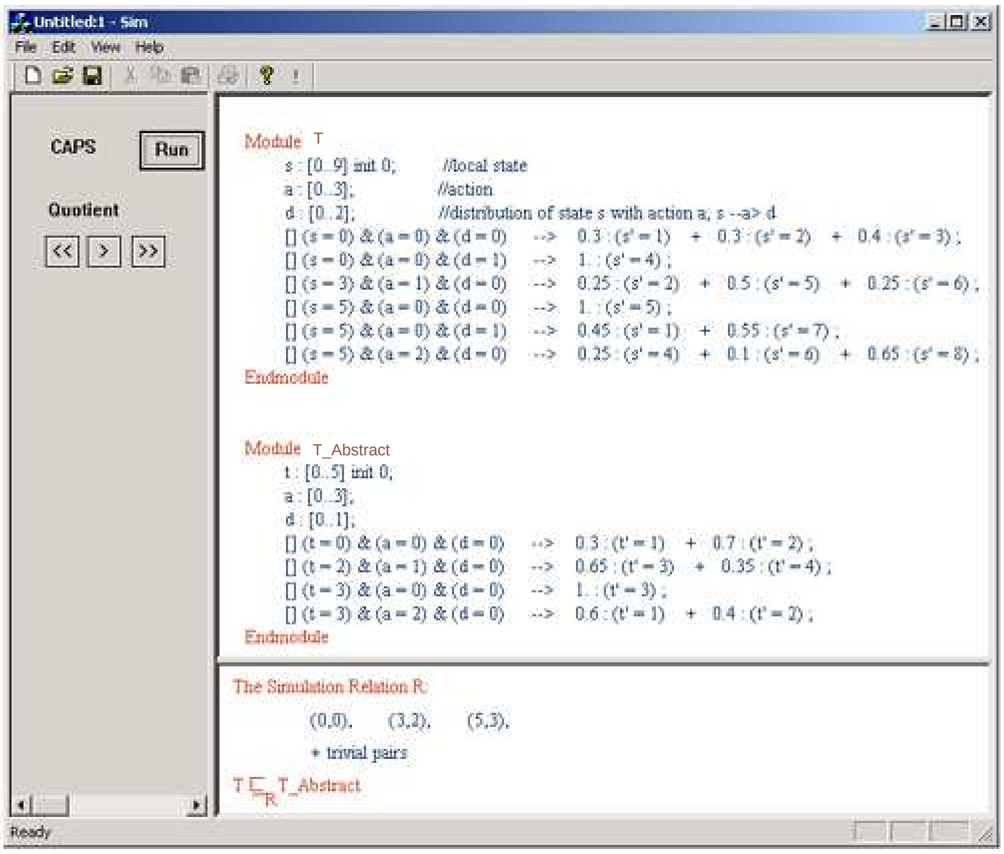
can transform with an action α. Each PTS begins with a negative order number, starting from 0.

```
0 // the first PTS
6 2 // number of states and number of actions

2 0 // The distribution matrix
0 0
0 0
0 1
0 0
0 0
0 0

0 0 // state 0 and action 0
0 0.25 0.25 0.5 0 0 0 // Distribution 1 of state 0 and action 0
0.5 0 0 0 0.5 0 0 // Distribution 2 of state 0 and action 0

3 1 // state 3 and action 1
0 0 0 0 0 0.5 0.5 // Distribution 1 of state 3 and action 1
-1 // a new PTS.
5 3
1 0 0
0 0 0
0 1 0
1 0 1
0 0 0
0 0
0 0.3 0.7 0 0
2 1
0 0 0 0.65 0.35
3 0
0 0 0 1 0
3 2
0 0.60 0.40 0 0
```

Figure 9.3: An input file to CAPS

9.3.2 Output Screen

The output screen for Compositional and Abstraction Checking is shown in Figure 9.4. The right frame is a Functional frame, where we can choose to perform either functinality of doing Quotient or Compositional and Abstraction checking in CAPS. The Main frame states two given PTS, the first is the original PTS T and the second is the abstract PTS $T_{Abstract}$. Below this is the Result frame. It shows the simulation relation R and states whether T is simulated by $T_{Abstract}$ or not.



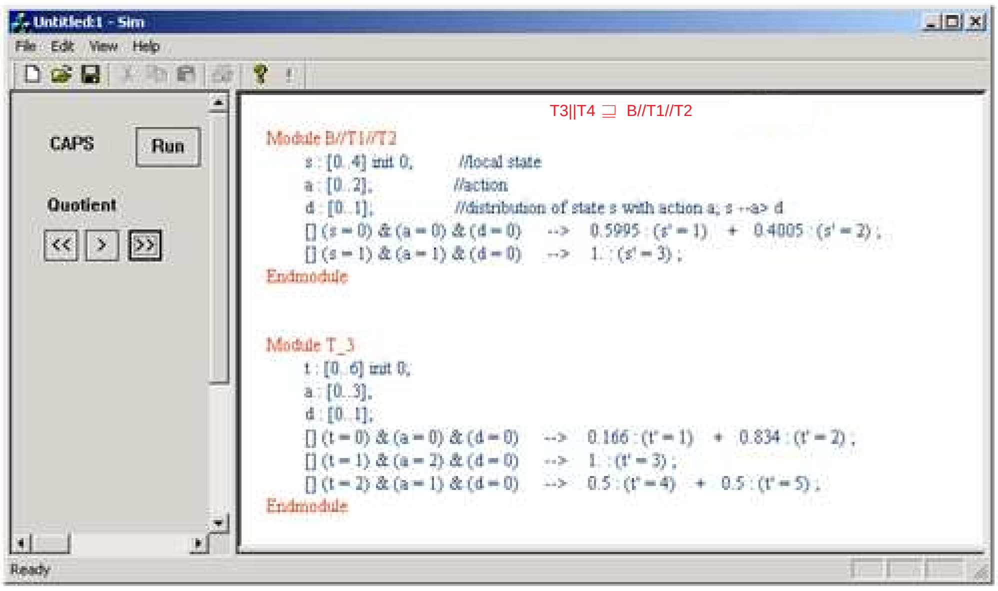
```
Module  T
      s : [0..9] init 0;          //local state
      a : [0..3];                 //action
      d : [0..2];                 //distribution of state s with action a; s --a> d
      [] (s = 0) & (a = 0) & (d = 0)    -->   0.3 : (s' = 1)   +  0.3 : (s' = 2)   +  0.4 : (s' = 3) ;
      [] (s = 0) & (a = 0) & (d = 1)    -->   1. : (s' = 4) ;
      [] (s = 3) & (a = 1) & (d = 0)    -->   0.25 : (s' = 2)  +  0.5 : (s' = 5)   +  0.25 : (s' = 6) ;
      [] (s = 5) & (a = 0) & (d = 0)    -->   1. : (s' = 5) ;
      [] (s = 5) & (a = 0) & (d = 1)    -->   0.45 : (s' = 1)  +  0.55 : (s' = 7) ;
      [] (s = 5) & (a = 2) & (d = 0)    -->   0.25 : (s' = 4)  +  0.1 : (s' = 6)   +  0.65 : (s' = 8) ;
Endmodule


Module  T_Abstract
      t : [0..5] init 0;
      a : [0..3];
      d : [0..1];
      [] (t = 0) & (a = 0) & (d = 0)    -->   0.3 : (t' = 1)   +  0.7 : (t' = 2) ;
      [] (t = 2) & (a = 1) & (d = 0)    -->   0.65 : (t' = 3)  +  0.35 : (t' = 4) ;
      [] (t = 3) & (a = 0) & (d = 0)    -->   1. : (t' = 3) ;
      [] (t = 3) & (a = 2) & (d = 0)    -->   0.6 : (t' = 1)   +  0.4 : (t' = 2) ;
Endmodule
```

```
The Simulation Relation R:

          (0,0),     (3,2),     (5,3),

          + trivial pairs
```

$T \sqsubseteq_R T\_Abstract$

Figure 9.4: The interface of the output result for Compositional and Abstraction Checking

The output screen for Quotient Checking is shown in Figure 9.5. In the right frame, you can either perform step by step factoring out individual PTS component to the specification B by pressing on the button ">>" or run through the whole parallel system by pressing on the button ">". The main frame shows the quotient structure of the blocking PTS $B//T_i$ and the PTS T_i.



T3||T4 ⊒ B//T1//T2

```
Module B//T1//T2
      s : [0..4] init 0;        //local state
      a : [0..2];               //action
      d : [0..1];               //distribution of state s with action a; s --a> d
      [] (s = 0) & (a = 0) & (d = 0)    -->   0.5995 : (s' = 1)   +   0.4005 : (s' = 2) ;
      [] (s = 1) & (a = 1) & (d = 0)    -->   1. : (s' = 3) ;
Endmodule


Module T_3
      t : [0..6] init 0;
      a : [0..3];
      d : [0..1];
      [] (t = 0) & (a = 0) & (d = 0)    -->   0.166 : (t' = 1)   +   0.834 : (t' = 2) ;
      [] (t = 1) & (a = 2) & (d = 0)    -->   1. : (t' = 3) ;
      [] (t = 2) & (a = 1) & (d = 0)    -->   0.5 : (t' = 4)   +   0.5 : (t' = 5) ;
Endmodule
```

Figure 9.5: The interface of the output result for the Quotient Checking

Chapter 10

Experimental Result

In this chapter, we discuss the results obtained by our implementation acronymed CAPS. For the Compositional and Abstract Checking, several tests are experimented, including some examples we have discussed in this thesis such as the parallel exams, the die performance. For the Quotient Checking, we did not use any concrete example, however we use suitable test cases created by the tester, which exposed some of the error of the implementation and showed the effectiveness of the quotient technique. Empirical measurement of time and space usage can always be verified later.

10.1 Compositional and Abstract Test Cases

Test 1: The die

The idea of this test case is extended from the test case Dice Programs of PRISM [49]. This case study considers two PTS, one is the original PTS T and one is the abstraction $T_{Abstract}$. Figure 10.1 models a die using only fair coins. Starting at the root state s_0, one repeatedly tosses a coin. Every time heads appears, one takes the upper branch and when tails appears, the lower branch. This continues until the value of the die is decided. There is only one action in this test case therefore we omit any symbol of action on all transitions out from each state. The abstract is constructed by applying partitioning technique for a specific reachability property, in this case eventually the value of the dice is six.

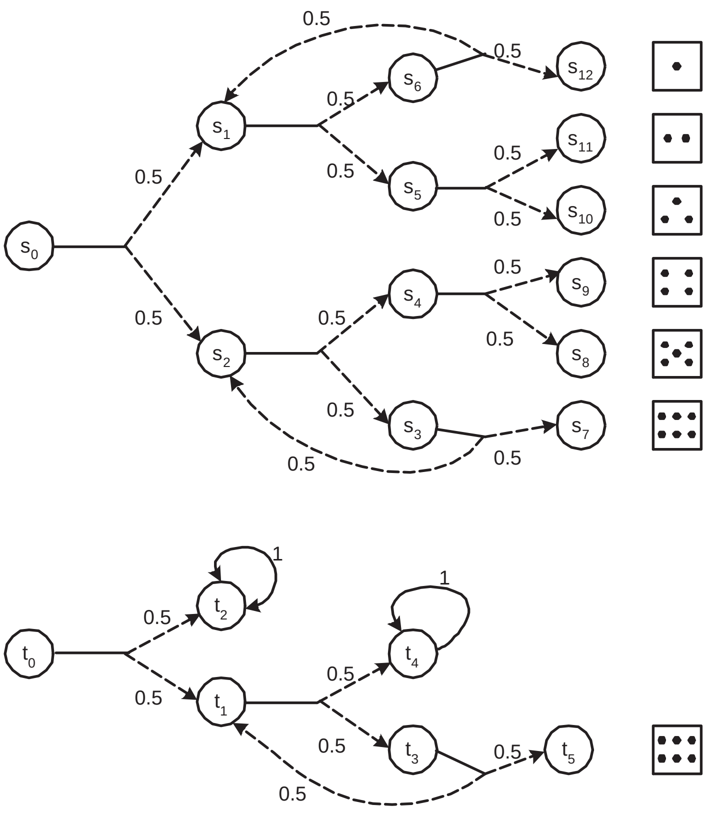
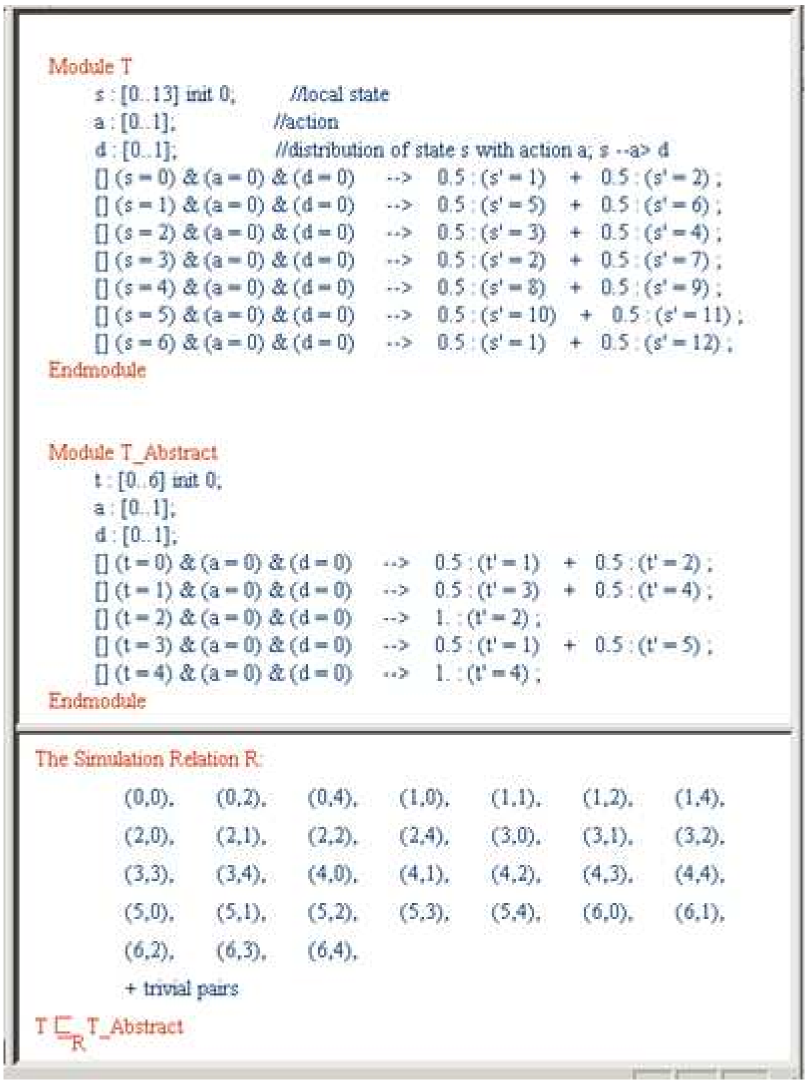


Figure 10.1: The die with an abstract

The result obtained by CAPS is shown in Figure 10.2:



```
Module T
      s : [0..13] init 0;        //local state
      a : [0..1];               //action
      d : [0..1];               //distribution of state s with action a; s --a> d
      [] (s = 0) & (a = 0) & (d = 0)    -->   0.5 : (s' = 1)   +   0.5 : (s' = 2) ;
      [] (s = 1) & (a = 0) & (d = 0)    -->   0.5 : (s' = 5)   +   0.5 : (s' = 6) ;
      [] (s = 2) & (a = 0) & (d = 0)    -->   0.5 : (s' = 3)   +   0.5 : (s' = 4) ;
      [] (s = 3) & (a = 0) & (d = 0)    -->   0.5 : (s' = 2)   +   0.5 : (s' = 7) ;
      [] (s = 4) & (a = 0) & (d = 0)    -->   0.5 : (s' = 8)   +   0.5 : (s' = 9) ;
      [] (s = 5) & (a = 0) & (d = 0)    -->   0.5 : (s' = 10)   +   0.5 : (s' = 11) ;
      [] (s = 6) & (a = 0) & (d = 0)    -->   0.5 : (s' = 1)   +   0.5 : (s' = 12) ;
Endmodule


Module T_Abstract
      t : [0..6] init 0;
      a : [0..1];
      d : [0..1];
      [] (t = 0) & (a = 0) & (d = 0)    -->   0.5 : (t' = 1)   +   0.5 : (t' = 2) ;
      [] (t = 1) & (a = 0) & (d = 0)    -->   0.5 : (t' = 3)   +   0.5 : (t' = 4) ;
      [] (t = 2) & (a = 0) & (d = 0)    -->   1. : (t' = 2) ;
      [] (t = 3) & (a = 0) & (d = 0)    -->   0.5 : (t' = 1)   +   0.5 : (t' = 5) ;
      [] (t = 4) & (a = 0) & (d = 0)    -->   1. : (t' = 4) ;
Endmodule
```

```
The Simulation Relation R:
          (0,0),    (0,2),    (0,4),    (1,0),    (1,1),    (1,2),    (1,4),
          (2,0),    (2,1),    (2,2),    (2,4),    (3,0),    (3,1),    (3,2),
          (3,3),    (3,4),    (4,0),    (4,1),    (4,2),    (4,3),    (4,4),
          (5,0),    (5,1),    (5,2),    (5,3),    (5,4),    (6,0),    (6,1),
          (6,2),    (6,3),    (6,4),
          + trivial pairs
```

$T \sqsubseteq_R T\_Abstract$

Figure 10.2: The output screen of the Dice example

Obviously, the pair of the two initial states $(s_0, s_1) \in R$. Therefore we conclude that $T \sqsubseteq T_{Abstract}$.

Test 2: The parallel exams

In this test case, we recall the example of the parallel exams in section 2.2.2. Assume a student is going to attend the exams of his two courses. We are interested in asking that whether this student passes both of these courses with probability at least 0.8 with in three trials. Instead of checking a complex probabilistic system T we may examine the question in an abstract of T, which is definitely smaller. The Figure 10.3 models a complete parallel exams system and a simple abstract.

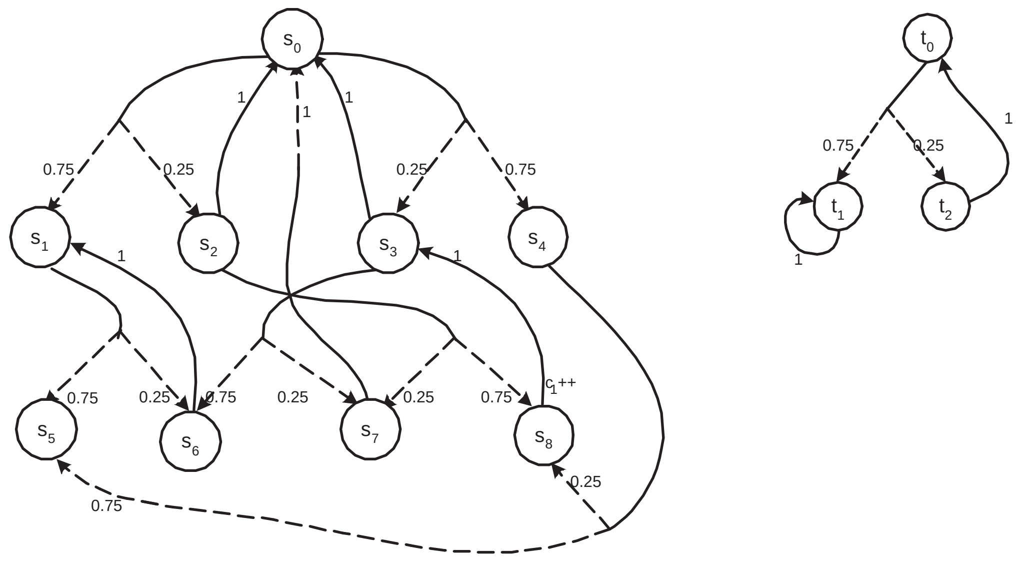
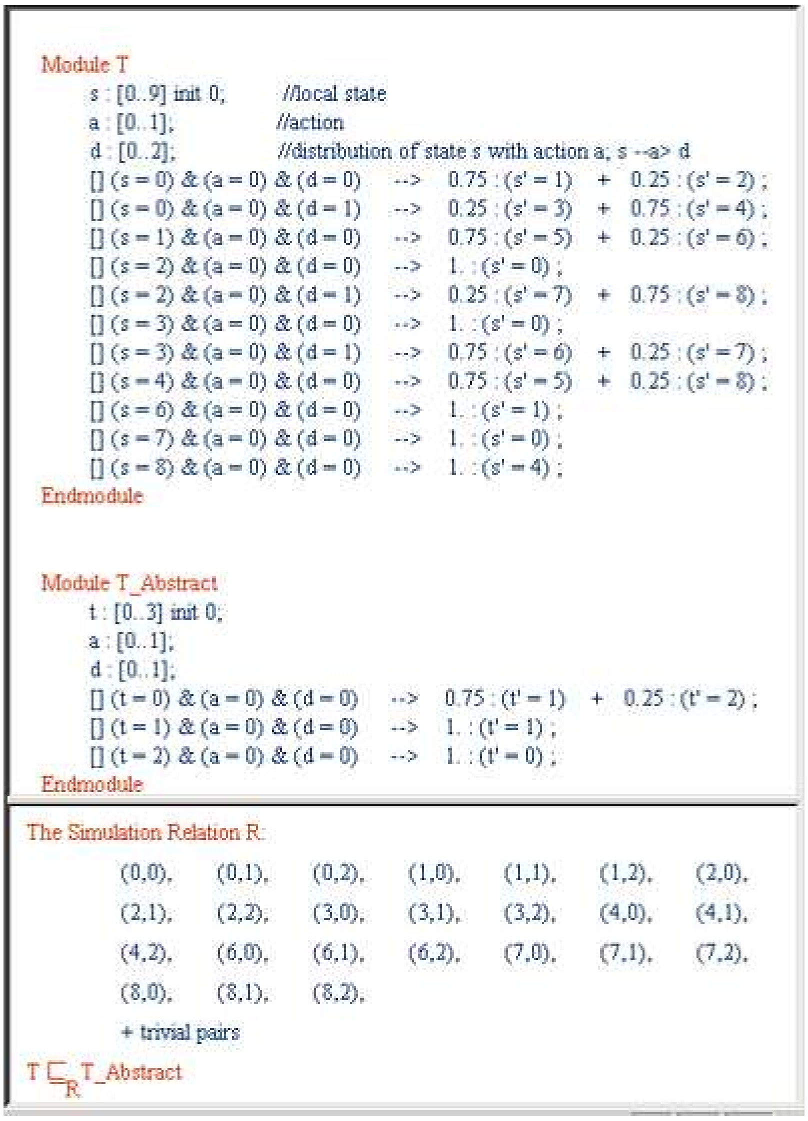


Figure 10.3: Composition of two PTS exams and an abstract

The result obtained by CAPS is shown in Figure 10.4:



```
Module T
      s : [0..9] init 0;        //local state
      a : [0..1];               //action
      d : [0..2];               //distribution of state s with action a; s --a> d
      [] (s = 0) & (a = 0) & (d = 0)    -->   0.75 : (s' = 1)   +   0.25 : (s' = 2) ;
      [] (s = 0) & (a = 0) & (d = 1)    -->   0.25 : (s' = 3)   +   0.75 : (s' = 4) ;
      [] (s = 1) & (a = 0) & (d = 0)    -->   0.75 : (s' = 5)   +   0.25 : (s' = 6) ;
      [] (s = 2) & (a = 0) & (d = 0)    -->   1. : (s' = 0) ;
      [] (s = 2) & (a = 0) & (d = 1)    -->   0.25 : (s' = 7)   +   0.75 : (s' = 8) ;
      [] (s = 3) & (a = 0) & (d = 0)    -->   1. : (s' = 0) ;
      [] (s = 3) & (a = 0) & (d = 1)    -->   0.75 : (s' = 6)   +   0.25 : (s' = 7) ;
      [] (s = 4) & (a = 0) & (d = 0)    -->   0.75 : (s' = 5)   +   0.25 : (s' = 8) ;
      [] (s = 6) & (a = 0) & (d = 0)    -->   1. : (s' = 1) ;
      [] (s = 7) & (a = 0) & (d = 0)    -->   1. : (s' = 0) ;
      [] (s = 8) & (a = 0) & (d = 0)    -->   1. : (s' = 4) ;
Endmodule


Module T_Abstract
      t : [0..3] init 0;
      a : [0..1];
      d : [0..1];
      [] (t = 0) & (a = 0) & (d = 0)    -->   0.75 : (t' = 1)   +   0.25 : (t' = 2) ;
      [] (t = 1) & (a = 0) & (d = 0)    -->   1. : (t' = 1) ;
      [] (t = 2) & (a = 0) & (d = 0)    -->   1. : (t' = 0) ;
Endmodule

The Simulation Relation R:
         (0,0),   (0,1),   (0,2),   (1,0),   (1,1),   (1,2),   (2,0),
         (2,1),   (2,2),   (3,0),   (3,1),   (3,2),   (4,0),   (4,1),
         (4,2),   (6,0),   (6,1),   (6,2),   (7,0),   (7,1),   (7,2),
         (8,0),   (8,1),   (8,2),
         + trivial pairs
T ⊑_R T_Abstract
```

Figure 10.4: The output screen of the parallel exams

Obviously, the pair of the two initial states $(s_0, t_0) \in R$. Therefore we conclude that $T \sqsubseteq T_{Abstract}$.

10.2 Quotient Test Cases

Test 3:

The test case is shown in Figure 10.6. Consider two PTSs T_1 and T_2 and the blocking PTS B, we wish to know that if the parallel system $T_1||T_2 \sqsupseteq B$ without having to construct the complete parallel system. By applying the quotient algorithm and runing the test with CAPS, we have obtained the result in Figure **??**. Obviously, in the end of the quotient process, all the components T_i's are removed from the parallel system and the specification B is simultaneously transformed with respect to ($\sqsupseteq$), see Figure 10.6, 10.7 and 10.8 . Therefore we conclude that $T_1||T_2 \sqsupseteq B$

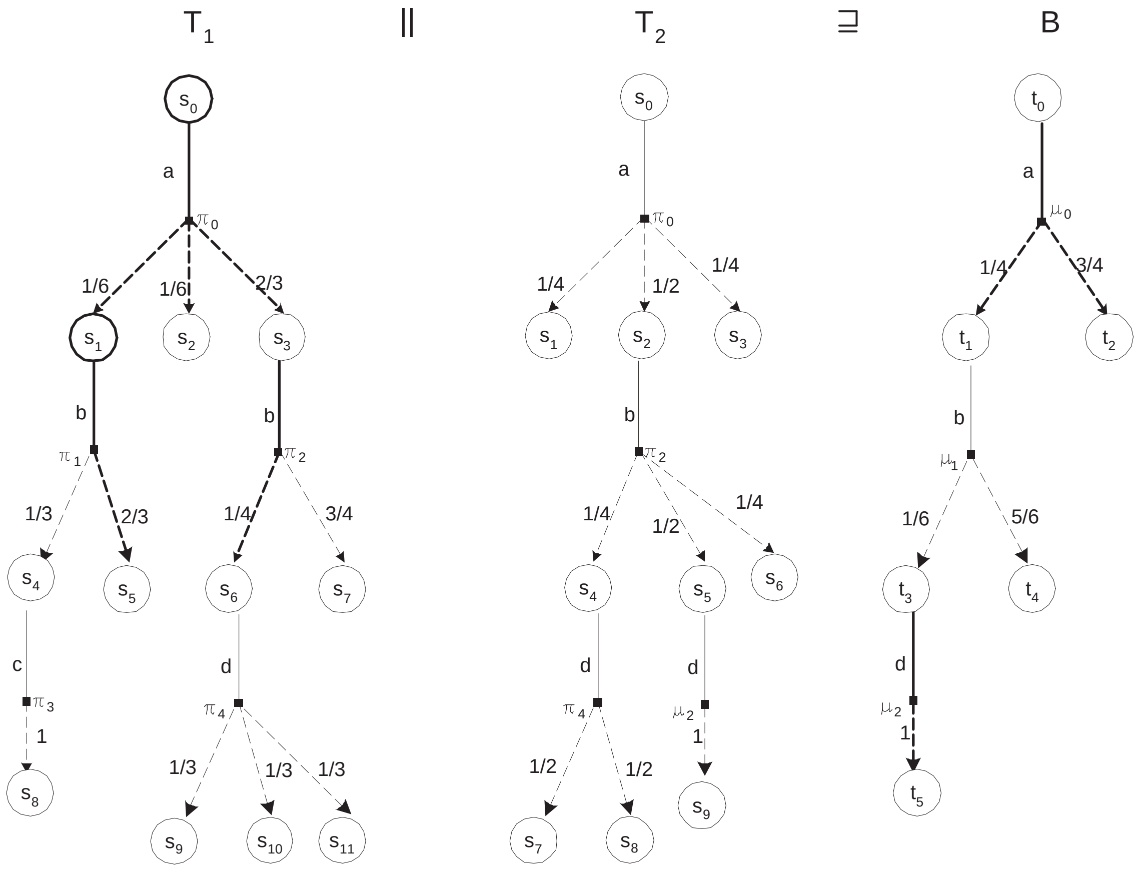
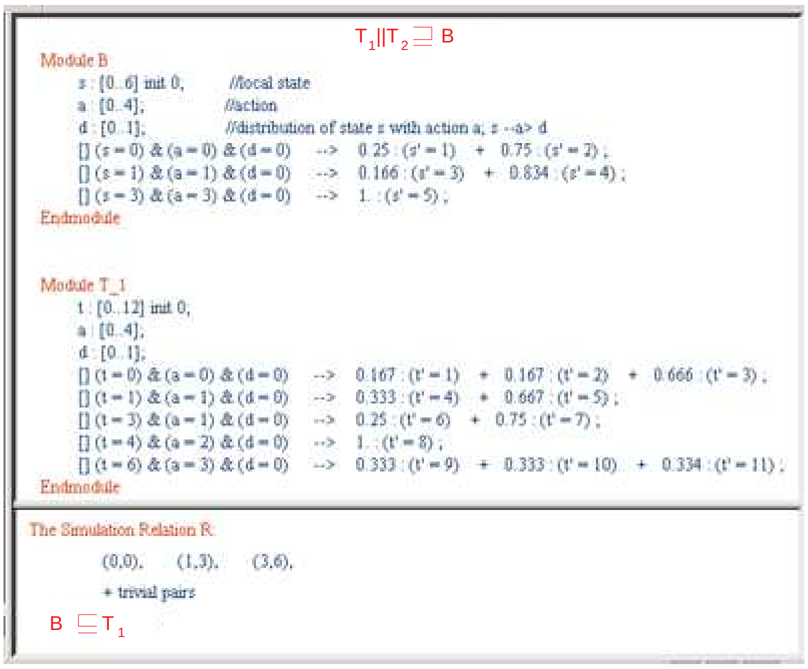


Figure 10.5: Two PTSs in parallel with an abstract blocking PTS

In this figure, the algorithm prepares to construct the quotient $B//T_1$. However the simulation relation of B and T_1 is first computed.



$T_1||T_2 \sqsupseteq B$

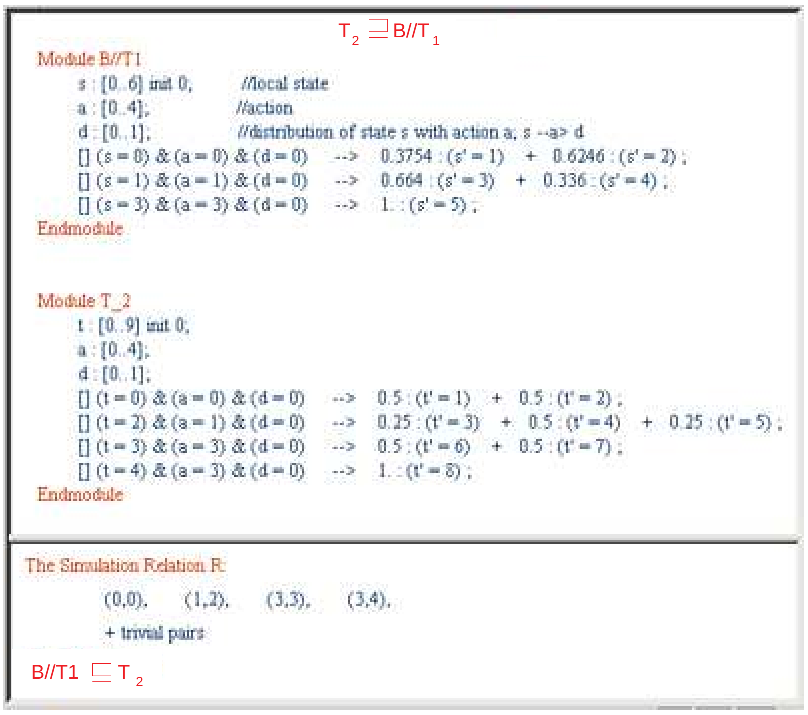
```
Module B
      s : [0..6] init 0,          //local state
      a : [0..4];                 //action
      d : [0..1];                 //distribution of state s with action a; s --a> d
      [] (s = 0) & (a = 0) & (d = 0)    -->   0.25 : (s' = 1)   +   0.75 : (s' = 2) ;
      [] (s = 1) & (a = 1) & (d = 0)    -->   0.166 : (s' = 3)   +   0.834 : (s' = 4) ;
      [] (s = 3) & (a = 3) & (d = 0)    -->   1. : (s' = 5) ;
Endmodule


Module T_1
      t : [0..12] init 0,
      a : [0..4];
      d : [0..1];
      [] (t = 0) & (a = 0) & (d = 0)    -->   0.167 : (t' = 1)   +   0.167 : (t' = 2)   +   0.666 : (t' = 3) ;
      [] (t = 1) & (a = 1) & (d = 0)    -->   0.333 : (t' = 4)   +   0.667 : (t' = 5) ;
      [] (t = 3) & (a = 1) & (d = 0)    -->   0.25 : (t' = 6)   +   0.75 : (t' = 7) ;
      [] (t = 4) & (a = 2) & (d = 0)    -->   1. : (t' = 8) ;
      [] (t = 6) & (a = 3) & (d = 0)    -->   0.333 : (t' = 9)   +   0.333 : (t' = 10)   +   0.334 : (t' = 11) ;
Endmodule
```

```
The Simulation Relation R:

          (0,0),     (1,3),     (3,6),

          + trivial pairs
```

$B \sqsubseteq T_1$

Figure 10.6: The initial output screen

The quotient $B//T_1$ has been constructed in this figure. Morever, T_1 is also removed from the parallel system. The algorithm is ready for next step.

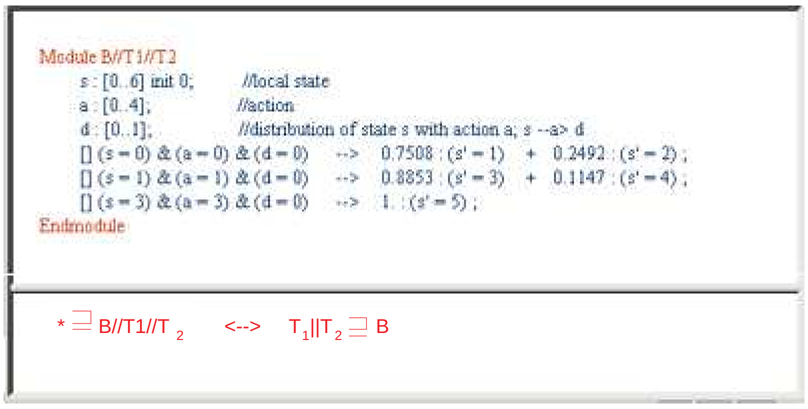


```
Module B//T1
      s : [0..6] init 0,         //local state
      a : [0..4];               //action
      d : [0..1];               //distribution of state s with action a; s --a> d
      [] (s = 0) & (a = 0) & (d = 0)    -->   0.3754 : (s' = 1)   +   0.6246 : (s' = 2) ;
      [] (s = 1) & (a = 1) & (d = 0)    -->   0.664 : (s' = 3)   +   0.336 : (s' = 4) ;
      [] (s = 3) & (a = 3) & (d = 0)    -->   1. : (s' = 5) ;
Endmodule


Module T_2
      t : [0..9] init 0,
      a : [0..4];
      d : [0..1];
      [] (t = 0) & (a = 0) & (d = 0)    -->   0.5 : (t' = 1)   +   0.5 : (t' = 2) ;
      [] (t = 2) & (a = 1) & (d = 0)    -->   0.25 : (t' = 3)   +   0.5 : (t' = 4)   +   0.25 : (t' = 5) ;
      [] (t = 3) & (a = 3) & (d = 0)    -->   0.5 : (t' = 6)   +   0.5 : (t' = 7) ;
      [] (t = 4) & (a = 3) & (d = 0)    -->   1. : (t' = 8) ;
Endmodule
```

```
The Simulation Relation R:
         (0,0),     (1,2),     (3,3),     (3,4),
         + trivial pairs
```

Figure 10.7: The output screen of constructin $B//T_1$

Finally, we obtained the quotient $((B//T_1)//T_2)$, that means the quotient process is perfomred successfully. We conclude that $T_1||T_2 \sqsupseteq B$



```
Module B//T1//T2
      s : [0..6] init 0;         //local state
      a : [0..4];               //action
      d : [0..1];               //distribution of state s with action a; s --a> d
      [] (s = 0) & (a = 0) & (d = 0)    -->   0.7508 : (s' = 1)   +   0.2492 : (s' = 2) ;
      [] (s = 1) & (a = 1) & (d = 0)    -->   0.8853 : (s' = 3)   +   0.1147 : (s' = 4) ;
      [] (s = 3) & (a = 3) & (d = 0)    -->   1. : (s' = 5) ;
Endmodule
```

Figure 10.8: The output screen of constructing $(B//T_1)//T_2$

Chapter 11

Conclusion

11.1 Conclusion

The aims of this thesis was to avoid the state explosion problem in probabilistic models by using simulation based abstractions. This led to the development of the tool CAPS which was based on establishing good abstracts for components in an asynchronous parallel composition of a model. The main bottleneck of this tool was that it is very user-dependent, requiring the user to input both the model and the abstraction.

The remainder of this thesis started with the aim of seeking to eliminate this drawback by automatically generating these abstractions. This goal however evolved and this report is the result.

In our previous work, we only examined an interleaving (asynchronous) parallel composition. We avoided the state space explosion by seeking abstracts of components and then constructing the parallel composition with these abstracts. It was therefore interesting to consider and study the avoidance of this parallel composition by rather quotienting. We develop further the theories of The Quotienting Technique, for PTSs, and extend a Minimization Algorithm to probabilistic transition systems. This led to transforming our specifications from logic-based to an appropriate transition-based specification. Usually, the transformed specification grows in size and requires a simplification heuristic to keep the size under control. The Minimization algorithm for PTSs was intended to be the simplification heuristic. However, we realized that, after quotienting, any structure obtained was already minimal and could therefore not be reduced or minimized further. This led us to propose that instead of minimizing (the transformed specification) after quotienting, we rather minimize the components of the system (or clusters of the components), before starting with the Quotienting technique.

Categorically, we conclude on the building blocks of this thesis.

Probabilistic Transition Systems

We have examined the probabilistic labelled transition system as our main modelling system. We have further considered its parallel composition in terms of the synchronous, asynchronous and the interleaving mixed sort parallel composition. We have extended all our abstraction methods to it and described ways to generate reduced PTSs structures. These methods are based on the Probabilistic Simulation preorder and equivalence relation. We also described a specification model based on PTSs called the Blocking Probabilistic Transitions System which is a structural version of Probabilistic Modal Logic.

Compositional Abstraction

We have described a basis for which components of an asynchronous parallel composition can be abstracted individually or together with other components. The abstraction of the components are then used in place of the original concrete components. We further explored ways of computing the simulation preorder between two PTSs and used algorithms for computing the maximum flow in networks to establish the preorder and conclude if the two PTS simulate each other.

The Quotient Technique

We started by exploring the Quotienting Technique for finite state systems and the use of linear inequation solving to generate the quotient structure. We then extended this to our probabilistic transition system. We developed a specification formalism for this technique, the blocking PTS, which explicitly portrayed our reachability properties. We have also considered various parallel compositionalities and developed algorithms to generate their quotient structures.

By this approach, our our whole model checking problem is reduced to that of generating the quotienting structure, which offers us a means of abstraction.

Minimization

We have discussed an algorithm which allows us to systematically generate a reduced PTS structure. This can be used to minimize a parallel composition as input and output a smaller reduced structure with respect to simulation equivalence. We had hoped to use this algorithm to simplify and minimize the transformed specification in the quotienting technique. However, alternatively we propose to use this algorithm on components before quotienting and not afterwards. This will also help to keep the size of the transformed specification in check.

The Implementation

We first developed a tool as an implementation the algorithms that we proposed for the methods of compositionality and abstraction based on the Simulation Preorder, acronymed CAPS (Compositional Abstraction for Probabilistic Systems). We further extended this tool by implementing the algorithms for the Quotienting Technique, for our specification. We realized that the quotient structure had infact, exactly the same structure as the blocking PTS of the specification. Hence the main issue, was computing the exact probabilities of the distributions of the quotient structure. Although, we did not gather any empirical evidence to support a hypothesis that the methodologies discussed, show a significant saving on time and space in the verification of these models, an intuitive argument should exhibit this fact.

11.2 Further Work

There is still the consideration of automatically generating abstracts of components of a model as was originally considered. Although this will be more user-friendlier and more effective, it is not easy to solve in general. We have suggested solutions in our specific case to obtaining abstractions for components in model checking.

In quotienting, we considered the interesting properties of reachability with the specific model of blocking PTSs. A possible extension will be to consider a broader spectrum of properties, where safety as well as liveness properties are verified.

As by our minimization, an extension could consider bisimulation based minimization which is strongly preserving.

Appendix

Please note the meanings of these symbols when reading this thesis.

$\sqsubseteq$: simulation relation

$\preceq$: simulation preorder

$\equiv$: simulation equivalence

$\bowtie$: a general relation between a model and another model as specification. Could be $\sqsubseteq$.

$\rceil\lceil$: generalization of the parallel composition

$//$: quotienting operator

$\leq$ is usually for size, as in less than or equal to. Sometimes also for the simulation relation.

The Greek letters should be read within the context they are stated.

$(\alpha, \beta, \gamma, \delta, \mu, \rho, \pi)$ usually denote a distribution over states.

(α, β) sometimes denote actions in the Act set.

a, b, c usually represent actions.

$(\alpha_1, \alpha_2, \ldots)$ denote equivalence classes.

(ψ, ϕ) usually represent specification properties.

Bibliography

[1] *Ahiable A., Hoang T.*, Compositional Abstraction by Probabilistic Simulation

[2] *Ravindra K. Ahuja, James B. Orlin, Clifford Stein and Robert E. Tarjan*, Improved Algorithms for Bipartite Network Flow

[3] *Henrik R. Andersen*, Partial Model Checking

[4] *C. Baier, J. Katoen, H. Hermanns, B. Haverkort*, Simulation for Continous-Time Markov Chains

[5] *Christel Baier, Bettina Engelen, and Mila Majster-Cederbaum*, Deciding Bisimilarity and Similarity for Probabilistic Processes

[6] *Christel Baier*, On Algorithmic Verification Methods for Probabilistic Systems

[7] *Christel Baier*, Polynomial Time Algorithms for Testing Probabilistic Bisimulation and Simulation

[8] *Christel Baier, Edmund N. Clarke, Vasiliki Hartonas-Garmhausen, Marta Kwiatkowska and Mark Ryan*, Symbolic Model Checking for Probabilistic Processes

[9] *Bodentien N. O., Poulsen L. O.*, The quotient verification technique applied to State/Event Systems

[10] *Bodentien N. O., Vestergaard J., Friis J., Kristoffersen K. J., Larsen K. G.*, Verification of Large State/Event Systems by Quotienting

[11] *Anders Borjesson, Kim G. Larsen and Arne Skou*, Generality in Design and Compositional Verification in TAV

[12] *A. Bianco and L. de Alfaro*, Model Checking of Probabilistic and Nondetermistic Systems

[13] *M. Brown, E. Clarke, O. Grumberg*, Characterizing finite Kripke Structures in Propositional Temporal Logic

[14] *R. E. Bryant*, Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers.

[15] *Doron Bustan, Orna Grumberg and David E. Long*, Model Checking and Abstraction

[16] *D. Bustan, O. Grumberg*, Simulation Based Minimization. In the 17th International Conference on Automated Deduction (CADE'00), Pittsburgh, June 2000.

[17] *Edmund N. Clarke, Orna Grumberg and David E. Long*, Model Checking and Abstraction

[18] *E. M. Clarke, E. A. Emerson and A. P. Sistla*, Automatic Verification of Finite-State Concurrent Systems using Temporal Logics Specification: A practical Approach.

[19] *E. M. Clarke, E. A. Emerson* , Design and Synthesis of synchronization skeletons using branching time temporal logic.

[20] *Pedro R. D'Argenio, Bertrand Jeannet, Henrik E. Jensen and Kim G. Larsen*, Reachability Analysis of Probabilistic Systems by Successive Refinements

[21] *Pedro R. D'Argenio, Bertrand Jeannet, Henrik E. Jensen and Kim G. Larsen*, Reduction and Refinement Strategies for Probabilistic Analysis

[22] *Rob J. van Glabeek, Scott A. Smolka and Bernhard Steffen*, Reactive, Generative, and Stratefied Models of Probabilistic Processes

[23] *van Glabbeek R. J., Smolka S. A., Steffen B., Tofts C. M. N.*, Reactive, Generative, Stratified Models of Probabilistic Processes

[24] *Godefroid P.*, Partial-Order methods for the Verification of Concurrent Systems: An Approach to the State Explosion Problem

[25] *Hans Hansson and Bengt Jonsson*, A Logic About Reasoning about Time and Reliability

[26] *A. Harding, M. Ryan, P. -Y. Schobbens*, Approximating ATL* in ATL

[27] *Y. Hsieh, S. P. Levitan* , Model Abstraction for Formal Verification

[28] *M. Huth* , Possibilistic and Probabilistic Abstraction-Based Model Checking

[29] *Bengt Jonsson and Kim G. Larsen*, Specification and Refinement of Probabilistic Processes

[30] *Bengt Jonsson, Wang Yi and Kim G. Larsen*, Probabilistic Extensions of Process Algebra

[31] *B. Jonsson*, Simulation between Specifications of Distributed Systems

[32] *Jou C., Smolka S. A.*, Equivalences, congruences, and a complete axiomatizations for probabilistic processes.

[33] *Joost-Pieter Katoen*, Concepts, Algorithms and Model Checking

[34] *R. P. Kurshan* , Formal Verification of Coordinating Processes

[35] *Francois Laroussinie, Kim G. Larsen*, Compositional Model Cheking of Real Time Systems

[36] *Kim G. Larsen and Arne Skou*, Bisimulation through Testing

[37] *Kim G. Larsen and Bent Thomsen*, Partial Specifications and Compositional Verification

[38] *Kim G. Larsen*, Context-Dependent Bisimulation Between Processes. PhD thesis, University of Edinburgh, Mayfield Road, Edinburgh, Scotland, 1986.

[39] *Kim G. Larsen, Arne Skou* Compositional Verification of Probabilistic Processes

[40] *K. L. McMillan*, Symbolic Model Checking: An approach to the State Explosion Problem.

[41] *R. Milner*, Communication and Concurrency

[42] *Lind-Nielsen J., Andersen H. R, Behrmann G., Hulgaard H., Kristoffersen K. J., Larsen K. G.*, Verification of Large State/Event Systems using Compositionality and Dependency Analysis.

[43] *Peled D.*, All from One, One from All: On Model Checking Using Representatives

[44] *R. Segala and N. A. Lynch*, Probabilistic Simulation for Probabilistic Processes

[45] *R. Segala*, Modelling and Verification of Randomized Distributed Real-Time Systems

[46] *Vestergaard J.*,The Quotienting Technique for Probabilistic Systems

[47] *Liu Xinxin, Kim G. Larsen*, Equation Solving Using Modal Transition Systems

[48] http://www.irisa.fr/prive/bjeannet/prob/prob.html

[49] http://www.cs.bham.ac.uk/ dxp/prism/