

Compositional Abstraction and Refinement of Probabilistic Automata

Master Student

Falak Sher

Reg No. 286082

Supervisors: Prof. Dr. Ir. Joost-Pieter Katoen
Prof. Lakemeyer, Ph.D
Daniel Klink, Ph.D

Contents

1	Introduction	3
1.1	Related Work	3
1.2	Contributions	4
1.3	Structure of the Thesis	5
2	Preliminaries	7
2.1	Notations	7
2.2	Stochastic Processes	8
2.3	Discrete-time Models	9
2.3.1	Discrete-time Markov Chains	9
2.3.2	Discrete-time Markov Decision Processes (DTMDPs)	10
2.4	Probabilistic Simulation and Bisimulation	11
2.4.1	Probabilistic Bisimulation	11
2.4.2	Probabilistic Simulation	12
3	Compositional Modeling for Non-Probabilistic Systems	15
3.1	Labeled Transition Systems (LTS)	16
3.2	Modal Transition Systems (MTS)	16
3.2.1	Satisfaction	18
3.2.2	Abstraction	18
3.2.3	Refinement	20
3.2.4	Conjunction	21
3.2.5	Parallel Composition	22
4	Compositional Modeling for Probabilistic Systems	25
4.1	Probabilistic Automata (PA)	26
4.2	Abstract Probabilistic Automata (APA)	27
4.3	Abstract Constraint Probabilistic Automata (ACPA)	29
4.3.1	Satisfaction	31
4.3.2	Consistency	32
4.3.3	Abstraction	34
4.3.4	Refinement	37
4.3.5	Conjunction	38
4.3.6	Parallel Composition	40
4.3.7	Valuation Synchronization	44

CONTENTS

5 Conclusion	47
A Proofs of Theorems	49
Bibliography	69

1

Introduction

We are passing through an era where every aspect of our lives is directly or indirectly influenced by computer systems. Our safety, medical treatments, education, etc, are heavily dependent on computer systems. And with the advancement in science and technology, these systems are getting more complex day by day. For example, an X-Ray device being used these days is much more complex and sophisticated as compared to the one used fifty years ago. Therefore, an utmost care is required in the design and development of such systems as a minute flaw in a critical system such as a nuclear reactor may result in catastrophic events with human casualties and lasting damage to the environment. It is also evident that designing a monolithic model of a system is one of the most difficult and error prone tasks. Therefore, the best way is to break down a system into its constituent components and design each one of them separately. These components can, then, be plugged together to get the composite model of a system.

In this thesis we try to lay down foundations of a compositional design methodology for *probabilistic systems* - a system where time is assumed to be discrete. We propose a new abstraction, abstract constraint probabilistic automata, and use it to construct a specification theory for probabilistic automata; that relates the notions of specification and implementation with a satisfaction or a refinement relation, and provides a set of operators - composition and conjunction - that together supports stepwise design of a system.

1.1 Related Work

In this section we briefly review what has already been done in the area of compositional modeling of probabilistic systems. In case of non-probabilistic systems, the theory of modal transition systems (MTS) [HJS01] provides a useful specification formalism supporting refinement as well as conjunction and parallel composition of specifications. Generalizing the notion of modal transition systems for compositional modeling of probabilistic systems, the formalism of interval Markov chains

(IMCs) was introduced in [JL91] with notions of satisfaction and refinement. Informally, an IMC extends the notion of Markov chains by having transitions labeled with intervals of allowed probabilities rather than individual probabilities. However, it has been proved by simple examples in [CDL⁺10] that the expressive power of IMCs is inadequate to support both conjunction and parallel composition of specifications. In another work [KKN09] interval Markov chains and modal transition systems are combined into abstract interactive Markov chains (AIMCs) in a natural and orthogonal way, and the correctness proofs for parallel and symmetric composition have been given. However, nothing has been discussed about the logical conjunction of AIMCs in this work.

The most relevant work related to compositional design methodology for probabilistic systems has been given in [CDL⁺10]. In this work a new abstraction, constraint Markov chains (CMCs), is introduced that is used as a tool in a stepwise compositional modeling of probabilistic systems. CMCs are a further extension of Markov chains allowing arbitrary constraints on the next-state probabilities from any state. CMCs are used to construct a specification theory for Markov chains that provides constructs for satisfaction, refinement, consistency checking, conjunction as well as parallel composition. In fact, in this thesis the work given in [CDL⁺10] has been extended for probabilistic automata with some additional features and results.

1.2 Contributions

In this section we give a brief comparison of the specification theory for Markov chains [CDL⁺10] with the one that we propose for probabilistic automata. We also highlight the additional features of our work that contribute to the compositional modeling of probabilistic systems.

- One of the limitations of (constraint) Markov chains (MCs) is that they do not allow parallel composition of asynchronously interacting components that may make nondeterministic and probabilistic choices. On the other hand, (abstract constraint) probabilistic automata (PA) have this power that gives an edge to our design methodology in this respect.
- Constraint Markov chains (CMCs) abstract Markov chains (MCs) in such a way that instead of having a concrete probability distribution from a state, a (possibly infinite) set of distributions are specified by constraining the next state probabilities. Similarly, each state in CMCs has a set of admissible valuations sets instead of a single concrete valuations set. However, an abstract constraint probabilistic automata (ACPA) goes a step further. It general-

izes probabilistic automata using notions of *must* and *may*-transitions as well as *must* and *may*-valuations as introduced for model transition systems (MTS). Moreover, like CMCs, in ACPA the target of every transition is specified by a set of (possibly infinite) distributions instead of a single concrete distribution. Thus, ACPA have all the features of MTSs as well as CMCs. Furthermore, the refinement relation for ACPAs is defined on the same lines as for MTSs and CMCs.

- In the specification theory for Markov chains we do not find any information about how to abstract CMCs. However, in our theory we describe the process of abstracting ACPAs in detail.
- Another major step forward is our work on compositional abstraction for ACPA. We propose to perform abstraction on the component level that can then be composed into a monolithic model. This technique is very helpful for reducing the peak memory consumption during the construction of the monolithic model.

1.3 Structure of the Thesis

In Chapter 2 we define notations that are used throughout the thesis and give some mathematical background. We also give a necessary background on stochastic processes as well as the definitions of discrete-time Markov chains (DTMCs) and discrete-time Markov decision processes (DTMDP). Finally, we present the concepts of simulation and bisimulation for DTMCs.

In Chapter 3 we give a necessary background on compositional design methodology for non-probabilistic systems using model transition systems. We describe the notions of satisfaction, abstraction, refinement as well as parallel composition for modal transition systems.

In Chapter 4 we describe our specification theory for abstract constraint probabilistic automata (ACPA) in detail. We extend the notions of satisfaction, abstraction, refinement as well as parallel composition for ACPA, given in Chapter 3. Moreover, we also describe the constructs for conjunction, consistency checking and valuation synchronizations - all indispensable ingredients for a compositional design methodology. At the end, in the appendix, we give formal proofs of all the theorems given in Chapter 4.

2

Preliminaries

In this chapter we define notations that are used throughout the thesis and give some mathematical background. Further, we recall the concept of stochastic processes as well as the definitions of discrete-time Markov chains (DTMCs) and discrete-time Markov decision processes (MDPs). Finally, we present the concepts of simulation and bisimulation for MDPs that would be helpful for understanding similar concepts later in this thesis.

2.1 Notations

Let X be a finite, non-empty set. A function f is a *distribution on X* iff $f : X \rightarrow [0, 1]$ and $f(X) = \sum_{x \in X} f(x) = 1$. For $A \subseteq X$, $f(A) = \sum_{x \in A} f(x)$. The support of distribution f is $\text{supp}(f) = \{x \in X \mid f(x) > 0\}$ and the set of all distributions on X is denoted by $\text{Dist}(X)$.

Let $f \in \text{Dist}(X)$ and $g \in \text{Dist}(Y)$. The point-wise product $(f \cdot g) : X \times Y \rightarrow [0, 1]$ is given as: $(f \cdot g)(x, y) = f(x) \cdot g(y)$ for $x \in X$, $y \in Y$, and $\sum_{(x,y) \in X \times Y} (f \cdot g)(x, y) = 1$. For $A \subseteq X$ and $B \subseteq Y$, $(f \cdot g)(A \times B) = \sum_{(x,y) \in A \times B} (f \cdot g)(x, y)$. Let a function $I : Y \times Y \rightarrow \{0, 1\}$ be given by $I(y, y') = 1$ iff $y = y'$; and for a set $B \subseteq Y$, $I(y, B) = 1$ iff $y \in B$. The function $I(y, \cdot)$ is given by $y \mapsto I(y, y')$ for all $y' \in Y$.

For sets W , X , Y and Z , let $f : W \rightarrow X$ and $g : Y \rightarrow Z$ be functions such that their Cartesian product $f \times g : W \times Y \rightarrow X \times Z$ is given as: $(f \times g)(w, y) = (f(w), g(y))$, where $w \in W$ and $y \in Y$.

Let X , Y and Z be finite, non-empty sets and let $R \subseteq X \times Y$ and $R' \subseteq Y \times Z$ be binary relations. The composition $R'' \subseteq X \times Z$ of these relations is denoted as $R \circ R'$. We use $\text{Dom}(R) = \{u \mid \exists v : (u, v) \in R\}$ to denote the domain, and $\text{Ran}(R) = \{v \mid \exists u : (u, v) \in R\}$ to denote the range of a binary relation R . For $x \in X$, let $R(x)$ denote the set $\{y \in Y \mid (x, y) \in R\}$; and similarly for

$y \in Y$, let $R^{-1}(y)$ denote the set $\{x \in X \mid (x, y) \in R\}$. For $A \subseteq X$, we define $R(A) = \cup_{x \in A} R(x)$; and for $B \subseteq Y$, we define $R^{-1}(B) = \cup_{y \in B} R^{-1}(y)$.

Three-valued truth domain. Let AP be a finite set of atomic propositions. A proposition is normally interpreted over the *two-valued truth domain* $\mathbb{B}_2 = \{\perp, \top\}$, which means that a proposition is either *true*, denoted by \top , or *false*, denoted by \perp . However, later on in thesis, we have situations in which we are not sure whether a proposition holds in a state or not. To handle these situations we resort to a three-valued truth domain.

The *three-valued truth domain* $\mathbb{B}_3 = \{\perp, ?, \top\}$ adds a new element $?$ to \mathbb{B}_2 which should be read as *don't know*. \mathbb{B}_3 forms a lattice with ordering $\perp < ? < \top$ and meet (\sqcap) and join (\sqcup) operations to get the minimum and the maximum of two elements respectively. Moreover, \mathbb{B}_3 turns into a *de Morgan* lattice when defining complementation \cdot^c such that \perp and \top are complementary to each other and $?^c = ?$.

For truth-values $x, y \in \mathbb{B}_3$, a binary operator \oplus is defined as:

$$x \oplus y = \begin{cases} x \sqcup y & \text{if } x \neq \perp \wedge y \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

2.2 Stochastic Processes

In this section we give a brief introduction of stochastic processes and some of their properties as far as they are needed in the context of this thesis. A comprehensive introduction of stochastic processes can be found in [Fel68].

A *stochastic process* is a collection of random variables $\{X(t) \mid t \in T\}$, defined on a probability space. The set T represents a time domain. In case of a discrete time domain, the set T is countable and normally we have $T = \mathbb{N}$. However, in case of a continuous time domain, the set T is uncountable and we have $T = \mathbb{R}_{\geq 0}$. The domain of $X(t)$ is called the state space of a stochastic process, and we assume that it is countable in the context of this thesis. A stochastic process has the *Markov property*, iff for all $t' \in T$:

$$Pr(\{X(t+t') = s' \mid \forall z \leq t : X(z) = s_z\}) = Pr(\{X(t+t') = s' \mid X(t) = s_t\})$$

Intuitively, the Markov property holds, if the probability of moving to a next state only depends on the current state and it does not depend on the history of the

process. Another important property that often is presumed is *time-homogeneity*. Formally, a stochastic process is time-homogeneous, iff for all $t, t' \in T$:

$$Pr(\{X(t + t') = s' \mid X(t) = s\}) = Pr(\{X(t') = s' \mid X(0) = s\})$$

This means that the subsequent states of the process do not depend on the time that has been passed since the process has been started.

2.3 Discrete-time Models

In this section we briefly recall discrete-time Markov chains (DTMCs), a fully probabilistic model, and discrete-time Markov decision processes (MDPs), a non-deterministic extension of DTMCs.

2.3.1 Discrete-time Markov Chains

A discrete-time Markov chain (DTMC) is a stochastic process with a possibly infinite, yet countable, set of states S . It has an initial distribution μ_0 that specifies a state $s \in S$ to be an initial state of the DTMC with probability $\mu_0(s)$. The probability to move in one step from state s to a *successor* state s' is given by a transition probability function $P(s, s')$, where $s, s' \in S$. We require DTMCs to be deadlock-free, i.e., each state has at least one successor state that is reachable with a positive probability. Moreover, each state may satisfy certain atomic propositions given by a mapping V . Formally, a DTMC is defined as follows:

Definition 2.3.1. (DTMC) A discrete-time Markov chain (DTMC) is a tuple (S, P, AP, V, μ_0) where:

- S is a finite, non-empty set of states,
- $P: S \times S \rightarrow [0, 1]$ is a transition probability function satisfying $P(s, S) = 1$ for all $s \in S$,
- AP is a finite, non-empty set of valuations,
- $V: S \times AP \rightarrow \mathbb{B}_2$ is a two-valued state-labeling function,
- $\mu_0 \in Dist(S)$ is an initial distribution

Example 2.3.1. Consider the discrete-time Markov chain M in Figure 2.1. It has three states s_0, s_1 and s_2 with s_0 as an initial state. From initial state there are two transitions: one with probability $1/3$ to state s_2 and the other with probability $3/4$ to state s_1 such that the total probability of leaving the initial state is 1. From each state s_1 and s_2 there is one transition with probability 1 to initial state.

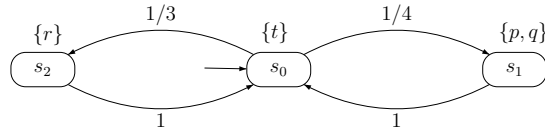


Figure 2.1: A Discrete Time Markov Chain (DTMC)

2.3.2 Discrete-time Markov Decision Processes (DTMDPs)

Like DTMCs, discrete-time Markov decision processes (MDPs) consist of an infinite set of states S , an initial distribution μ_0 and a labeling function V . However, a major difference between DTMCs and MDPs is that the behavior of MDPs is nondeterministic, i.e., in each state, one out of finitely many actions a can be chosen. Further, the target of every transition in MDPs is a probability distribution over the successor states. Thus, the probability to move in one step from a state s via action a to a successor state s' is given by the three-dimensional transition probability function $P(s, a, s')$. Note that in contrast to other popular models [Seg95, Her02], for all actions a there is at most one a -transition emanating a state in MDPs.

As for DTMCs, we require MDPs to be deadlock-free, i.e., for each state s there exists at least one action a such that $P(s, a, S) = 1$. Formally, an MDP is defined as follows:

Definition 2.3.2. (MDP) A discrete-time Markov decision process (MDP) is a tuple (S, A, P, AP, V, μ_0) where:

- S is a finite, non-empty set of states,
- $P: S \times A \times S \rightarrow [0, 1]$ is a transition probability function satisfying $P(s, a, S) \in [0, 1]$ for all $s \in S, a \in A$,
- AP is a finite, non-empty set of valuations,
- $V: S \times AP \rightarrow \mathbb{B}_2$ is a two-valued state-labeling function,
- $\mu_0 \in \text{Dist}(S)$ is an initial distribution

Example 2.3.2. Consider the MDP M in Figure 2.2. It has four states s_0, s_1, s_2 and s_3 with s_0 as an initial state having two enabled actions a and b . Note that when action a is selected, we get a probability distribution $(0, 3/4, 1/4, 0)$ over successor states. Moreover, in M for each a -transition we get only one probability distribution over successor states.

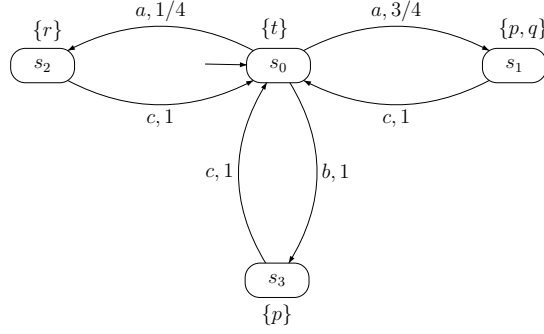


Figure 2.2: A Markov Decision Process (MDP)

2.4 Probabilistic Simulation and Bisimulation

In this section we recall the notions of simulation and bisimulation for probabilistic systems, and describe them, in particular, for MDPs. These concepts will be helpful in understanding the similar concepts later in this thesis.

2.4.1 Probabilistic Bisimulation

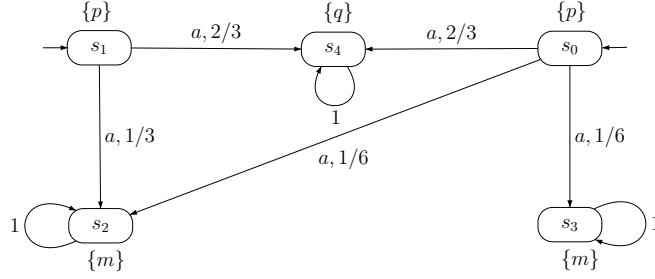
Bisimulation minimization is an important reduction method for all kinds of models from labeled transition systems (LTSs) to Markov decision processes (MDPs). In this process, we aggregate those states of a model that are similar to each other in all aspects, i.e., in terms of the labeling, the available nondeterministic choices and the successor states. In other words, states are bisimilar if they can mimic each others behavior in all possible ways. Formally, the bisimulation relation among states in an MDP is defined as follows:

Definition 2.4.1. (Bisimulation) Let $M = (S, P, AP, V, \mu_0)$ be an MDP. An equivalence relation $R \subseteq S \times S$ is a bisimulation relation, iff for all $a \in A$ and sRu , the following conditions hold:

- $V(s, p) = V(u, p)$ for all $p \in AP$, and
- $P(s, a, T) = P(u, a, T)$ for all $T \in S/R$,

where S/R denotes the quotient space under R . Then, R is a *bisimulation* relation and states $s, u \in S$ with sRu are bisimilar denoted as $s \approx u$.

Example 2.4.1. Consider the MDP $M = (S, P, AP, V, \mu_0)$ in Figure 2.3. In M the states s_0 and s_1 as well as s_2 and s_3 are bisimilar. Note that both s_0 and s_1 have the same labeling, p , and the transition probability via action a to each equivalent class


 Figure 2.3: $s_0 \approx s_1$ and $s_2 \approx s_3$.

in quotient space S/R is equal, i.e., $P(s_0, a, \{s_2, s_3\}) = 1/3 = P(s_1, a, \{s_2, s_3\})$ and $P(s_0, a, \{s_4\}) = 2/3 = P(s_1, a, \{s_4\})$.

2.4.2 Probabilistic Simulation

Simulation relations are preorders on a state space requiring that whenever state u simulates state s , denoted as $s \preceq u$, then u can mimic all stepwise behavior of s but in addition may also perform steps that cannot be matched by s . In the following, we give a definition of a simulation relation among distributions, and, then, use it in the definition of a simulation relation among states in an MDP.

Definition 2.4.2. Let S be a finite, non-empty set of states, and let $\mu \in \text{Dist}(S)$ and $\mu' \in \text{Dist}(S)$ be distributions on S . For a binary relation $R \subseteq S \times S$, μ is simulated by μ' w.r.t. R , denoted as $\mu \sqsubseteq_R \mu'$, iff there exists a weight function $\Delta : S \times S \rightarrow [0, 1]$ such that for all $u \in S$ and $v \in S$, the following conditions hold:

$$\begin{aligned} \Delta(u, v) > 0 &\Rightarrow uRv \\ \Delta(u, S) &= \mu(u) \\ \Delta(S, v) &= \mu'(v) \end{aligned}$$

In fact, \sqsubseteq_R is the lifting of relation R on states to distributions over states.

Definition 2.4.3. (Simulation) Let $M = (S, P, AP, V, \mu_0)$ be an MDP. Let $\mu, \mu' \in \text{Dist}(S)$. A binary relation $R \subseteq S \times S$ is a simulation relation, iff for all $a \in A$ and sRu , the following conditions hold:

- $V(s, p) = V(u, p)$ for all $p \in AP$, and
- $\forall \mu : s \xrightarrow{a} \mu \implies \exists \mu' : u \xrightarrow{a} \mu' \wedge \mu \sqsubseteq_R \mu'$

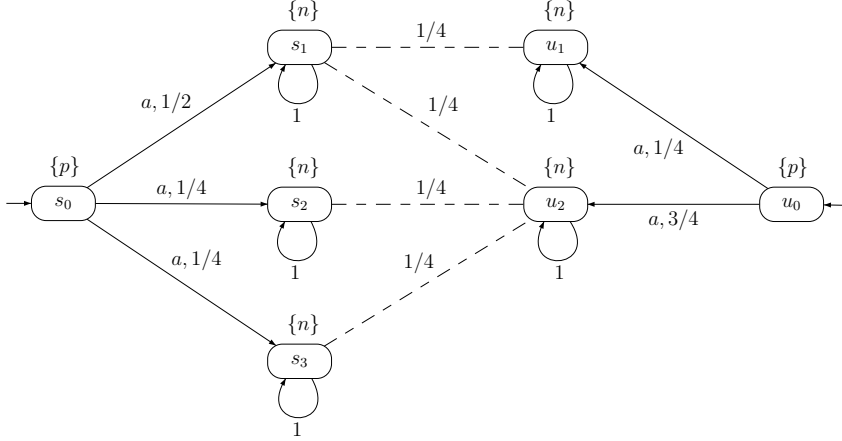


Figure 2.4: State s_0 is simulated by state u_0 , i.e., $s_0 \preceq u_0$

We write $s \preceq u$ iff sRu for some simulation relation R .

Example 2.4.2. Consider the MDP with a state space $S = \{s_0, s_1, s_2, s_3, u_0, u_1, u_2\}$ and two initial states s_0 and u_0 as depicted in Figure 2.4. Assume both initial states have an a -transition such that the distributions from s_0 and u_0 are μ and μ' respectively. We can see that the distribution μ is simulated by distribution μ' as there exists a weight function Δ that relates s_1 with u_1 and u_2 , s_2 with u_2 and s_3 with u_2 as indicated by dashed lines. Moreover, both s_0 and u_0 have same valuation p . Hence, s_0 is simulated by u_0 .

In the following chapter, we give an overview of compositional modeling for non-probabilistic systems. We briefly introduce the notions of satisfaction, refinement, abstraction as well as parallel composition for modal transition systems (MTSs). Late on in Chapter 4, we extend these notions for the abstract model, abstract constraint probabilistic automata (ACPA), that we propose to give a specification theory for compositional modeling of probabilistic systems.

3

Compositional Modeling for Non-Probabilistic Systems

In this chapter we discuss how functional behavior of a complex system can be modeled. By functional behavior, we refer to the capability of performing actions in a defined way. For example, “a client must first send a request before entering into receiving mode” may be a functional behavior of a client process. In compositional design methodology, we break down a system into sub-components and model each component separately. The functional behavior of each component is specified as a *Labeled Transition System (LTS)*, which contains all the states a component may reach and all the transitions it may perform. The composition of all these labeled transition systems yields the functional behavior of the whole system.

Moreover, we normally require the verification of some properties of a component (or a whole system), for which we may need an abstraction of its model specified by an LTS. The abstraction of an LTS can be represented as a *Modal Transition System (MTS)*. Thus, an MTS also represents an abstract behavior of an LTS and, hence, can be used to give the specification of a system at an abstract level.

We begin this chapter with a brief introduction of labeled transition systems and modal transition systems. After this we discuss how to perform abstraction and parallel composition on modal transition systems and how to relate the resulting models in terms of refinement. We also discuss the conjunction of two modal transition systems. For simplicity, we abstract from the internal details of a system and consider only its observable behavior. Moreover, we assume that a system has only one initial state.

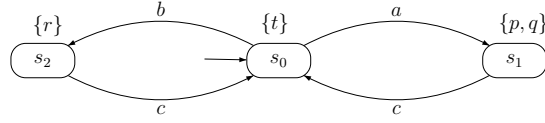


Figure 3.1: A Labeled Transition System (LTS)

3.1 Labeled Transition Systems (LTS)

The formal definition of a labeled transition system is given as:

Definition 3.1.1. (Labeled Transition System) A labeled transition system is a tuple (S, A, L, AP, V, s_0) where:

- S is a finite, non-empty set of states,
- A is a finite set of actions,
- $L: S \times A \times S \rightarrow \mathbb{B}_2$ is a *two-valued* transition function,
- AP is a finite set of valuations,
- $V: S \times AP \rightarrow \mathbb{B}_2$ is a *two-valued* state-labeling function, and
- $s_0 \in S$ is an initial state.

The labeling $L(s, a, s')$ identifies the “type” of a transition: \top and \perp indicate the presence and the absence of a transition respectively, where $s, s' \in S$. Similarly, the labeling $V(s, p)$ identifies the “type” of a valuation $p \in AP$.

Example 3.1.1. Consider the LTS $K = (S, A, L, AP, V, s_0)$ depicted in Figure 3.1. It has three states s_0, s_1 and s_2 having $\{t\}, \{p, q\}$ and $\{r\}$ as valuation sets respectively. State s_0 is an initial state of the system from which two actions a and b are enabled. On performing action a nondeterministically, the system goes to state s_1 , while on action b it goes to state s_2 . From both states s_1 and s_2 , the system goes back to the initial state on action c .

3.2 Modal Transition Systems (MTS)

A modal transition system is a generalization of a labeled transition system. In literature, an MTS is often defined as a pair of LTSs where one of them describes the optional (*may*) and the other describes the required (*must*) behavior of the

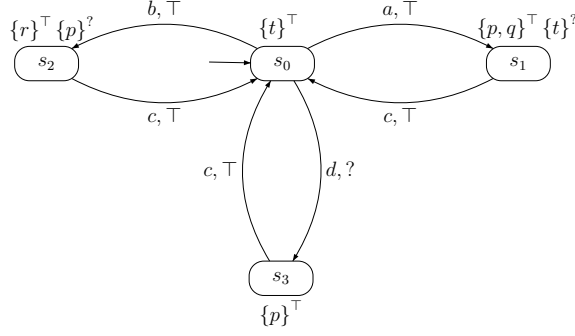


Figure 3.2: A Modal Transition System (MTS)

system. For example, the capability of transmitting messages repeatedly is a required behavior of a digital transmitter, whereas transmitting data at high rate (> 10 GBits/sec) could be an optional not a required behavior. In other words, an MTS gives the specification of a system, and it contains both the required and the optional behavior of a system. In the following we give another definition of an MTS that suits our needs in this thesis.

Definition 3.2.1. (Modal Transition System) A modal transition system is a tuple (S, A, L, AP, V, s_0) where:

- S is a finite, non-empty set of states,
- A is a finite set of actions,
- $L : S \times A \times S \longrightarrow \mathbb{B}_3$ is a *three*-valued transition function,
- AP is a finite set of valuations,
- $V : S \times AP \longrightarrow \mathbb{B}_3$ is a *three*-valued state-labeling function, and
- $s_0 \in S$ is an initial state.

The labeling $L(s, a, s')$ identifies the “type” of a transition: \top , $?$ and \perp indicate a *must*-transition, a *may*-transition and the absence of a transition respectively, where $s, s' \in S$, $a \in A$. Similarly, the labeling $V(s, p)$ identifies the “type” of a valuation, where $p \in AP$. Note that any LTS is an MTS in which all transitions are *must*-transitions and valuations of every state are *must*-valuations.

Example 3.2.1. Consider the MTS $M = (S, A, L, AP, V, s_0)$ depicted in Figure 3.2. It has four states s_0, s_1, s_2 and s_3 with s_0 as an initial state. The initial state s_0 has two *must*-transitions (s_0, a, s_1) and (s_0, b, s_2) , and one *may*-transition

(s_0, d, s_3) . Each other state has just one *must* c -transition that leads to the initial state. Note that the state s_1 has two set of valuations: $\{p, q\}$ is the *must*-valuations set while $\{t\}$ is the *may*-valuations set.

3.2.1 Satisfaction

An LTS that implements the required behavior of an MTS, and whose complete behavior is reflected in the MTS is said to be an implementation of the MTS. We formally define a relationship between an MTS and an LTS implementing it as:

Definition 3.2.2. (Satisfaction) Let $K = (S, A, L, AP, V, s_0)$ and $M = (S', A', L', AP', V', s'_0)$ be an LTS and an MTS respectively with $AP \subseteq AP'$ and $A \subseteq A'$. A relation $R \subseteq S \times S'$ is a satisfaction relation, iff for all sRs' , the following conditions hold:

- 1a. $\forall a \in A', \forall u' \in S' : L'(s', a, u') = \top \implies \exists u \in S : L(s, a, u) = \top \wedge uRu'$
- 1b. $\forall a \in A, \forall u \in S : L(s, a, u) = \top \implies \exists u' \in S' : L'(s', a, u') \neq \perp \wedge uRu'$
- 2a. $\forall p \in AP' : V'(s', p) = \top \implies V(s, p) = \top$
- 2b. $\forall p \in AP : V(s, p) = \top \implies V'(s', p) \neq \perp$

Item 1a requires that every *must*-transition from s' in M has a corresponding *must*-transition from s in K ; and Item 1b requires that every transition from s in K has a corresponding *must* or a *may*-transition from s' in M . Items 2a and 2b have a similar explanation for the proposition labeling.

We write $K \models M$ iff there exists a satisfaction relation between s_0 and s'_0 , and call K an *implementation* of M . The set of all implementations of M is given by $\llbracket M \rrbracket = \{K \mid K \models M\}$.

Example 3.2.2. The LTS K in Figure 3.1 satisfies the MTS M in Figure 3.2. $R = \{(s_0, s_0), (s_1, s_1), (s_2, s_2)\}$ is a satisfaction relation between K and M .

3.2.2 Abstraction

In this section we explain how an MTS is abstracted by partitioning its state space, i.e, by grouping sets of states into abstract ones. First we give definitions of *abstraction* and *concretization* functions that are used in the formal definition of abstraction of an MTS.

Definition 3.2.3. An abstraction function $\alpha : S \rightarrow S'$ is a surjective function that maps each element in state space S of a concrete model M to S' in the induced abstract model M' .



Figure 3.3: The MTS M (left) is abstracted by the MTS M' (right), i.e., $M' = \alpha(M)$

Definition 3.2.4. A concretization function $\gamma : S' \rightarrow 2^S$ is the inverse of abstraction function α . It returns the sets of concrete states in S that are represented by abstract ones in S' .

$\alpha(s)$ denotes the abstract state of state s and $\gamma(s')$ represents the set of all states that map to abstract state s' . As a result of abstraction of an MTS, we obtain a new MTS that not only covers all possible behaviors of the original MTS but may depict some additional behavior as well. A formal definition of abstraction is given as:

Definition 3.2.5. (Abstraction) For an MTS $M = (S, A, L, AP, V, s_0)$, the abstraction function $\alpha : S \rightarrow S'$ induces the MTS $\alpha(M) = (S', A, L', AP, V', s'_0)$, where for all $a \in A$ and $s', u' \in S'$:

$$L'(s', a, u') = \begin{cases} \top & \text{if } \forall s \in \gamma(s'), \exists u \in \gamma(u') : L(s, a, u) = \top \\ \perp & \text{if } \forall s \in \gamma(s'), \forall u \in \gamma(u') : L(s, a, u) = \perp \\ ? & \text{otherwise} \end{cases}$$

and for all $p \in AP$,

$$V'(s', p) = \begin{cases} \top & \text{if } \forall s \in \gamma(s') : V(s, p) = \top \\ \perp & \text{if } \forall s \in \gamma(s') : V(s, p) = \perp \\ ? & \text{otherwise} \end{cases}$$

Example 3.2.3. Consider the MTSs $M = (S, A, L, AP, V, s_0)$ (left) and $M' = (S', A, L', AP, V', s'_{02})$ (right) in Figure 3.3. Let the abstraction function $\alpha : S \rightarrow S'$ be given by $\alpha(s_0) = s'_{02} = \alpha(s_2)$, $\alpha(s_1) = s'_1$ and $\alpha(s_3) = s'_3$. As both s_0 and s_2 have a *must* a -transition to s_1 , therefore, s'_{02} has a *must* a -transition to $\alpha(s_1) = s'_1$.

Moreover, s'_{02} also has a *may* b -transition to s'_3 . It is of type *may* because only s_2 has a b -transition to $\gamma(s'_3)$ -states though it is of type *must*. Moreover, s'_{02} has a *must* p -valuation as both s_0 and s_2 have a *must* p -valuation. Note that q and t are *must*-valuations in s_0 and s_2 respectively, but they are *may*-valuations in s'_{02} because they are not common to s_0 and s_2 .

3.2.3 Refinement

In a stepwise design methodology we start the design of a system with a specification at an abstract level and then gradually elaborate it by adding details to it. During this process, we may also need to abstract a detailed specification of a component (or a system) for verification of some properties. In both cases we compare a specification either with its abstraction or its implementation. This is usually done using a *refinement* relation. Roughly speaking, if an MTS M refines an MTS M' , then any model of M is also a model of M' but not necessarily the converse. In the sequel, we discuss a *refinement* relation for MTSs.

Definition 3.2.6. (Refinement) Let $M = (S, A, L, AP, V, s_0)$ and $M' = (S', A', L', AP', V', s'_0)$ be two MTSs with $AP \subseteq AP'$ and $A \subseteq A'$. A relation $R \subseteq S \times S'$ is a refinement relation, iff for all sRs' , the following conditions hold:

- 1a. $\forall a \in A', \forall u' \in S' : L'(s', a, u') = \top \implies \exists u \in S : L(s, a, u) = \top \wedge uRu'$
- 1b. $\forall a \in A, \forall u \in S : L(s, a, u) \neq \perp \implies \exists u' \in S' : L'(s', a, u') \neq \perp \wedge uRu'$
- 2a. $\forall p \in AP' : V'(s', p) = \top \implies V(s, p) = \top$
- 2b. $\forall p \in AP : V(s, p) \neq \perp \implies V'(s', p) \neq \perp$

Item 1a requires that every *must*-transition of s' in M' must match some *must*-transition of s in M , i.e., the required behavior of s' must also be the required behavior of s . Item 1b requires that every *must* or *may*-transition of s in M must be reflected in s' in M' , i.e., the behavior of s must be derived from that of s' . Items 2a and 2b have a similar explanation for the proposition labeling.

Remark. A satisfaction relation is a special type of refinement relation.

We write $s \preceq s'$ iff sRs' for some refinement relation R , and $M \preceq M'$, for MTSs M and M' with initial states s_0 and s'_0 , iff $s_0 \preceq s'_0$. The relationship between an MTS and its abstraction is formalized by a refinement relation.

Theorem. [LT88] For an MTS M and abstraction function α :

$$M \preceq \alpha(M)$$

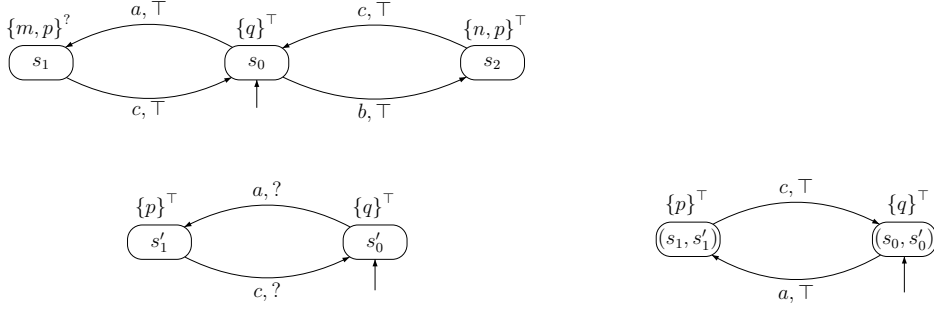


Figure 3.4: The MTS M (left top) and M' (left bottom) and their conjunction $M \wedge M'$ (right).

Example 3.2.4. Consider the MTSs M and M' given in Example 3.2.3. As M' is an abstraction of M , it follows $M \preceq M'$.

3.2.4 Conjunction

The *conjunction* operation is an important part of every specification theory. It is used to take the intersection of the requirements expressed by two or more specifications. The result of conjunction of two different MTSs is an MTS. Formally, it is given as:

Definition 3.2.7. (Conjunction) Let $M = (S, A, L, AP, V, s_0)$ and $M' = (S', A', L', AP', V', s'_0)$ be MTSs. The conjunction of M and M' , written as $M \wedge M'$, is given as:

$$M \wedge M' = (S \times S', A \cap A', \tilde{L}, AP \cap AP', \tilde{V}, (s_0, s'_0))$$

and for all $(s, s'), (u, u') \in S \times S'$ and $a \in A \cap A'$,

$$\tilde{L}((s, s'), a, (u, u')) = L(s, a, u) \oplus L'(s', a, u')$$

and for all $p \in AP \cap AP'$,

$$\tilde{V}((s, s'), p) = V(s, p) \oplus V'(s', p)$$

Example 3.2.5. Consider the MTSs $M = (S, A, L, AP, V, s_0)$ (left top) and $M' = (S', A', L', AP', V', s'_0)$ (left bottom) and their conjunction $M \wedge M' = (S \times S', A \cap A', \tilde{L}, AP \cap AP', \tilde{V}, (s_0, s'_0))$ (right) depicted in Figure 3.4. Both M and M' have an a -transition from their initial states s_0 and s'_0 but s_0 has a *must* a -transition

(s_0, a, s_1) whereas s'_0 has a *may* a -transition (s'_0, a, s'_1) ; therefore, the composite state (s_0, s'_0) has a *must* a -transition $((s_0, s'_0), a, (s_1, s'_1))$. Note that M' does not have a b -transition from its initial state, therefore, there is no b -transition from the composite state (s_0, s'_0) . Similar explanation goes for the proposition labeling in the composite states (s_0, s'_0) and (s_1, s'_1) .

3.2.5 Parallel Composition

Parallel composition is an operation in which several MTSs are combined into a single composite MTS that allows the interleaving of the actions of the component MTSs. This operation helps in compositional modeling of complex systems. The specification of each component is given by an independent MTS. These MTSs are then combined into a single composite MTS by parallel composition. In composite MTS each transition is either *synchronous* or *asynchronous*. In synchronous case all participating MTSs are required to perform their transitions simultaneously on some common action. Whereas in asynchronous case one of the participating MTSs performs a transition independently of others. Formally it is given as:

Definition 3.2.8. (Parallel Composition) Let $M = (S, A, L, AP, V, s_0)$ and $M' = (S', A', L', AP', V', s'_0)$ be MTSs with $AP \cap AP' = \emptyset$. The parallel composition of M and M' w.r.t. a *synchronization set* $\bar{A} \subseteq A \cap A'$ is given as:

$$M \parallel_{\bar{A}} M' = (S \times S', A \cup A', \tilde{L}, AP \cup AP', \tilde{V}, (s_0, s'_0))$$

where for all $(s, s'), (u, u') \in S \times S'$ and $a \in A \cup A'$:

$$\tilde{L}((s, s'), a, (u, u')) = \begin{cases} L(s, a, u) & \text{if } a \in A \setminus \bar{A} \\ L'(s', a, u') & \text{if } a \in A' \setminus \bar{A} \\ L(s, a, u) \sqcap L'(s', a, u') & \text{if } a \in \bar{A} \end{cases}$$

and for all $p \in AP \cup AP'$,

$$\tilde{V}((s, s'), p) = \begin{cases} V(s, p) & \text{if } p \in AP \\ V'(s', p) & \text{if } p \in AP' \end{cases}$$

For simplicity we assume that the set of valuations AP and AP' are disjoint; otherwise, we may exploit *three-valued* propositional labeling.

Example 3.2.6. Consider the MTSs $M = (S, A, L, AP, V, s_0)$ (left top) and $M' = (S', A', L', AP', V', s'_0)$ (left bottom) and their parallel composition $M \parallel_{\bar{A}} M' = (S \times$

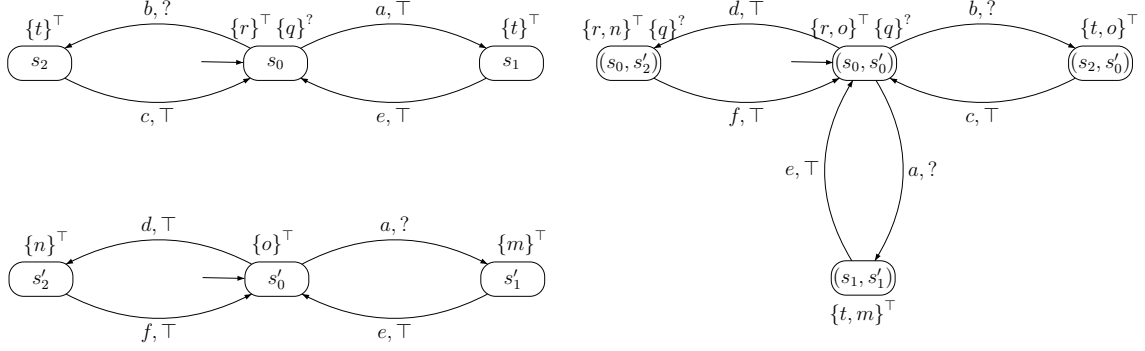


Figure 3.5: The MTSs M (left top) and M' (left bottom) and their parallel composition $M \parallel_{\bar{A}} M'$ w.r.t synchronization actions $\bar{A} = \{a, e\}$.

$S', A \cup A', \tilde{L}, AP \cup AP', \tilde{V}, (s_0, s'_0)$ (right) w.r.t. synchronization actions $\{a, e\} \subseteq A \cap A'$ depicted in Figure 3.5. Both M and M' synchronize on action a from their initial states s_0 and s'_0 but s_0 has a *must* a -transition whereas s'_0 has a *may* a -transition; therefore, the composite state (s_0, s'_0) has a *may* a -transition. Similarly, from states s_1 and s'_1 both M and M' synchronize on action e to their initial states. Moreover, the MTS M has a *may* b -transition from its initial state s_0 , that is depicted as a *may* b -transition from the composite state (s_0, s'_0) ; and the composite automaton moves asynchronously to state (s_2, s'_0) . The same is the case with a *must* d -transition from the initial state s'_0 in MTS M' . Furthermore, note that (s_0, s'_0) has two set of valuations: $\{r, o\}$ is a *must*-valuations set and $\{q\}$ is a *may*-valuations set, and they satisfy valuations of both constituent states s_0 and s'_0 .

It is proven that refinement is a precongruence with respect to parallel composition, thus, enabling compositional abstraction of MTSs.

Theorem. [LT88] *Refinement* \preceq is a precongruence w.r.t. $\parallel_{\bar{A}}$.

4

Compositional Modeling for Probabilistic Systems

In chapter 3 we discuss how functional behavior of a system is modeled in a stepwise design methodology. However, this design methodology is insufficient to capture all aspects of real systems, particularly, systems which have some probabilistic aspects. Therefore, while modeling functional behavior of such systems we are sometimes bound to make unrealistic assumptions. For example, when we specify that “a capability of transmitting messages repeatedly” is a required behavior of a digital transmitter, we make an assumption that the transmitter would never fail which is impossible to guarantee; and, therefore, in reality this behavior cannot be depicted by any model of the specification.

In addition to this, in functional design methodology we take the help of non-determinism to handle a situation where we cannot quantify the nature of a probabilistic event completely. For example, when we do not know the probability of a message loss or a processor failure exactly, we model it by nondeterminism. This is quite helpful in early design phases when a system is considered at a high level of abstraction and the information about the likelihood of an event is not known exactly; in later design stages the internal characteristics of a system become more dominant and, then, we can use probability values to resolve nondeterminism. In short, we cannot avoid probabilistic aspects of systems in their design.

For example, randomized algorithms such as leader election or consensus algorithms where coin-tossing experiments are used to break the symmetry between the processes such that, e.g., consensus is eventually reached with probability 1. Therefore, we need a design methodology that can handle probabilistic aspects of real systems.

In literature, there are different techniques to model probabilistic systems such as discrete-time Markov chains (DTMCs), Markov decision processes (MDPs),

probabilistic automata (PA), etc. In this thesis we select PA as a tool in compositional design methodology for probabilistic systems. We generalize PA to abstract probabilistic automata (APA) that are further generalized to abstract constraint probabilistic automata (ACPA). ACPA are then used to lay down the foundations of a specification theory for PA.

In this chapter we start with a brief introduction of PA. After this we explain how PA are generalized to APA and then to ACPA. Later on we discuss different operations on ACPA such as conjunction and parallel composition. We also discuss how to perform abstraction on an ACPA and the relation of the resulting models in terms of refinement.

4.1 Probabilistic Automata (PA)

Probabilistic automata provide a mathematical framework for modeling and analyzing probabilistic systems, in particular, systems of asynchronously interacting components that may make nondeterministic and probabilistic choices. PA are similar to nondeterministic automata but differ in that a transition leads to a probability distribution over a set of states instead of a single state. Formally, PA are given as:

Definition 4.1.1. (Probabilistic Automata) Probabilistic automata are tuples (S, A, L, AP, V, s_0) where:

- S is a finite, non-empty set of states,
- A is a finite set of actions,
- $L: S \times A \times \text{Dist}(S) \rightarrow \mathbb{B}_2$ is a *two-valued* distribution function,
- AP is a finite set of valuations,
- $V: S \times AP \rightarrow \mathbb{B}_2$ is a *two-valued* state-labeling function, and
- $s_0 \in S$ is an initial state.

We denote a transition relation (s, a, μ) as $s \xrightarrow{a} \mu$. Intuitively, on performing action a from state s , we get a probability distribution over a set of states that determines the next state of the system. We assume a finitely branching PA: for a state s , the number of pairs (a, μ) such that $s \xrightarrow{a} \mu$ is assumed to be finite. The labeling $L(s, a, \mu)$ identifies the “type” of a transition: \top indicates the presence and \perp indicates the absence of a transition. Similarly, the labeling $V(s, p)$ identifies the “type” of a valuation $p \in AP$. Moreover, we require PA to be deadlock-free:

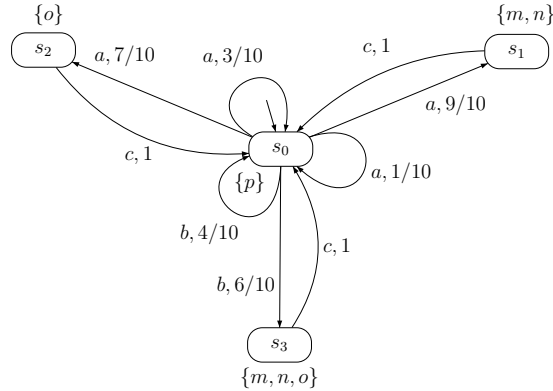


Figure 4.1: A Probabilistic Automaton (PA)

at least one action has to be enabled in each state, i.e., for each state s there exists an action a such that $P(s, a, S) = 1$. Furthermore, we assume that each state has a non-empty set of valuations. For simplicity, we assume a single initial state s_0 .

Example 4.1.1. Consider the PA $P = (S, A, L, AP, V, s_0)$ depicted in Figure 4.1. It has four states s_0, s_1, s_2 and s_3 having $\{p\}, \{m, n\}, \{o\}$ and $\{m, n, o\}$ as valuations sets respectively. On initial state s_0 two actions a and b are enabled, therefore, a nondeterministic choice is made to select one of them. When action a is selected two distributions $(3/10, 0, 7/10, 0)$ and $(1/10, 9/10, 0, 0)$ are available; therefore, again nondeterministically one of them is chosen and the system moves to the next state probabilistically. Whereas in case of action b only one distribution $(4/10, 0, 0, 6/10)$ exists that is used to decide the next state of the system. Each other state s_1, s_2 and s_3 has only one enabled action c , and the system moves to the initial state from it with probability 1.

4.2 Abstract Probabilistic Automata (APA)

In this section we explain how PA are abstracted by collapsing disjoint sets of concrete states into single abstract ones. The abstraction technique that we present is different from bisimulation quotienting in which only bisimilar states are grouped together, whereas in this technique any group of states can be chosen for abstraction. In fact, we use the concept of *must-* and *may-*transitions as well as *must-* and *may-*valuations as introduced for model transition systems (MTS). To abstract from the differences in the states' available nondeterministic choices of transitions (s, a, μ) for a given action a , we apply the concept of *must-* and *may-*transitions.

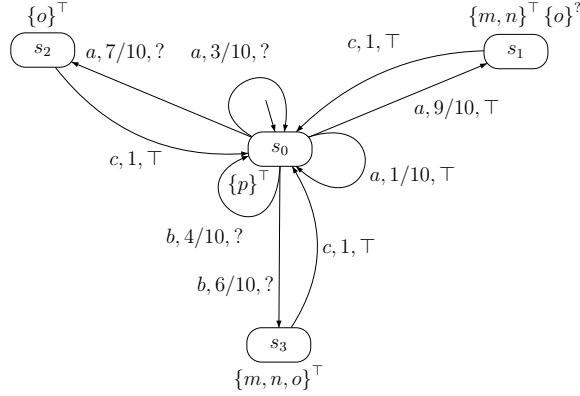


Figure 4.2: An Abstract Probabilistic Automaton (APA)

Similarly, for the differences in valuations among states, we use the concept of *must*- and *may*-valuations. As a result of this we get:

Definition 4.2.1. (Abstract PA) Abstract probabilistic automata are tuples (S, A, L, AP, V, s_0) where:

- S is a finite, non-empty set of states,
- A is a finite set of actions,
- $L : S \times A \times \text{Dist}(S) \longrightarrow \mathbb{B}_3$ is a *three*-valued transition function,
- AP is a finite set of valuations,
- $V : S \times AP \longrightarrow \mathbb{B}_3$ is a *three*-valued state-labeling function, and
- $s_0 \in S$ is an initial state.

The labeling $L(s, a, \mu)$ identifies the “type” of a transition: \top , $?$ and \perp indicate a *must*-transition, a *may*-transition and the absence of a transition respectively. Similarly, the labeling $V(s, p)$ identifies the “type” of a valuation $p \in AP$. Note that any PA is an APA in which all transitions are *must*-transitions and valuations of every state are *must*-valuations.

Example 4.2.1. Consider the APA $M = (S, A, L, AP, V, s_0)$ depicted in Figure 4.2. It has four states s_0, s_1, s_2 and s_3 each having a *must*-valuations set $\{p\}^\top, \{m, n\}^\top, \{o\}^\top$ and $\{m, n, o\}^\top$ respectively, whereas s_1 also has a *may*-valuations set $\{o\}^?$. Note that from initial state s_0 two *a*-transitions exists, one is a *may*-transition with distribution $(3/10, 0, 7/10, 0)$ whereas the other is a *must*-transition

with distribution $(1/10, 9/10, 0, 0)$. In case of action b only one *must*-transition exists with distribution $(4/10, 0, 0, 6/10)$. Each other state s_1 , s_2 and s_3 has only one *must* c -transition with distribution $(1, 0, 0, 0)$.

4.3 Abstract Constraint Probabilistic Automata (ACPA)

We go a step further and introduce ACPA that are a generalization of APA. We base our work on constraint Markov chains (CMCs) [CDL⁺10] that are an extension of DTMCs. In contrast to DTMCs, where each state has a probability distribution over successor states, in CMCs we do not have such concrete distributions. Instead, the distribution from each state s is given by a set of variables, each of them, say x_u , represents the probability value of a transition from state s to state u . As the domain of each such variable is $[0, 1]$, we use arithmetic expressions and/or inequalities to bound the values of these variables in such a way that the rules of probability theory are satisfied, i.e. $\sum_{u \in S} x_u = 1$.

Definition 4.3.1. (Constraint) An arithmetic expression or an inequality that bounds the values of the variables, it contains, is called a constraint.

A constraint specifies the allowed values for a group of variables it contains. Therefore, each state s in CMCs may allow multiple (possibly infinite) distributions over successor states; and each such distribution is an evaluation of all the variables that satisfies the constraints given for the state s .

Definition 4.3.2. (Constraint Function) The conjunction of all constraints given for a state in CMC is called the constraint function of the state.

A constraint function, denoted as φ , is a finite representation of possibly infinite set of distributions allowed by a state. It is modeled by a *characteristic function*, which for a given probability distribution evaluates to *true* iff the probability values of the distribution satisfy all of its constraints.

We extend the above mentioned technique for APA, in which each state has a *must*/*may* set of distributions for an action a . For each such set of distributions we design a constraint function that, along with accepting these distributions, has the tightest possible bounds for its constituent constraints. As each constraint function represents a set of distributions on the state space S of APA, the set of all constraint functions defined on S is given as $C(S)$. As a result of this step APA are generalized to ACPA. Formally, ACPA are given as:

Definition 4.3.3. (Abstract Constraint Probabilistic Automaton) Abstract constraint probabilistic automata are tuples (S, A, L, AP, V, s_0) where:

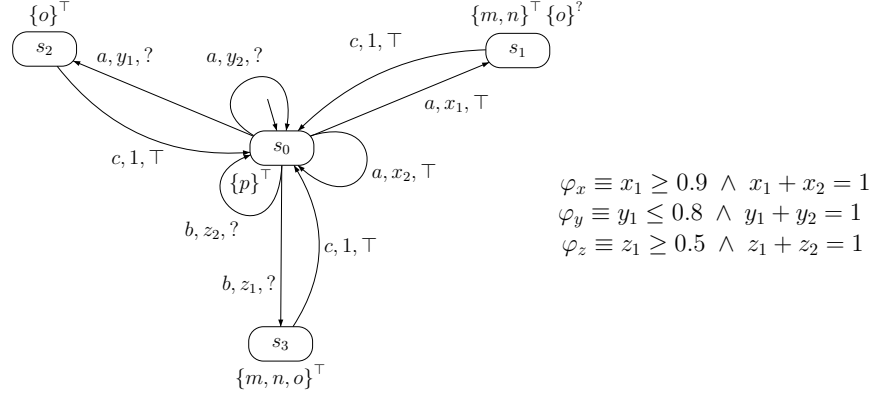


Figure 4.3: An Abstract Constraint Probabilistic Automaton (ACPA)

- S is a finite, non-empty set of states,
- A is a finite set of actions,
- $L : S \times A \times C(S) \longrightarrow \mathbb{B}_3$ is a *three*-valued state-constraint function,
- AP is a set of valuations,
- $V : S \times AP \longrightarrow \mathbb{B}_3$ is a *three*-valued state-labeling function, and
- $s_0 \in S$ is an initial state.

The labelings $L(s, a, \varphi)$ and $V(s, p)$ identify the “type” of a constraint function $\varphi \in C$ and an valuation $p \in AP$ respectively, similar to the ones given for APA. Note that every APA is an ACPA in which for each state s , every *must*-*may*-transition (s, a, μ) can be represented by a *must*-*may*-transition (s, a, φ) such that the constraint function φ only accepts one distribution namely μ .

In the definition of ACPA like CMCs [CDL⁺10], no particular type of constraints is implied for constraint functions. There are several classes of constraints that can be used in the definition of constraint functions: interval, linear or polynomial that have increasing expressiveness.

Example 4.3.1. Consider the ACPA $N = (S, A, C, L, AP, V, s_0)$ as depicted in Figure 4.3. It is similar to the APA M given in Figure 4.2 but differs in one aspect: in M the target of every transition from s_0 is a concrete distribution, whereas in N it is a constraint function that, in fact, can represent an infinite number of concrete distributions. Note that N has a *must* a -transition (s_0, a, φ_x) , a *may* a -transition (s_0, a, φ_y) , and a *may* b -transition (s_0, b, φ_z) from initial state s_0 . Each of

these transitions represents a possibly infinite number of transitions each having a concrete distribution. For example, the *must a*-transition $(s_0, a, (1/10, 9/10, 0, 0))$ in the APA M is represented by the *must a*-transition (s_0, a, φ_x) in the ACPA N . In short, the ACPA N represents infinitely many APAs and the APA M is one such example.

Let $Sat(\varphi)$ be an infinite set of distributions that are satisfied by a constraint function φ . The satisfaction set of the product of two constraint functions φ and φ' is given as: $Sat(\varphi \cdot \varphi') = Sat(\varphi) \cdot Sat(\varphi') = \{\mu \cdot \mu' \mid \mu \in Sat(\varphi), \mu' \in Sat(\varphi')\}$. Similarly, the satisfaction set of $\varphi \cdot I(x, \cdot)$ is given as: $Sat(\varphi \cdot I(x, \cdot)) = \{\mu \cdot I(x, \cdot) \mid \mu \in Sat(\varphi)\}$. Let $Act(s)$ be a set of actions that are enabled in a state $s \in S$.

4.3.1 Satisfaction

We use Definition 3.2.2 of satisfaction for an LTS and an MTS, and extend it for an ACPA and a PA that implements it. Formally, it is given as:

Definition 4.3.4. (Satisfaction) Let $P = (S, A, L, AP, V, s_0)$ and $N = (S', A', L', AP', V', s'_0)$ be a PA and an ACPA respectively with $AP \subseteq AP'$ and $A \subseteq A'$. A relation $R \subseteq S \times S'$ is a satisfaction relation, iff for all sRs' , the following conditions hold:

- 1a. $\forall a \in A', \forall \varphi' \in C(S') : L'(s', a, \varphi') = \top$
 $\implies \exists \mu \in Dist(S) : L(s, a, \mu) = \top \wedge \mu \sqsubseteq_R \mu' \text{ for some } \mu' \in Sat(\varphi')$
- 1b. $\forall a \in A, \forall \mu \in Dist(S) : L(s, a, \mu) = \top$
 $\implies \exists \varphi' \in C(S') : L'(s', a, \varphi') \neq \perp \wedge \mu \sqsubseteq_R \mu' \text{ for some } \mu' \in Sat(\varphi')$
- 2a. $\forall p \in AP' : V'(s', p) = \top \implies V(s, p) = \top$
- 2b. $\forall p \in AP : V(s, p) = \top \implies V'(s', p) \neq \perp$

Intuitively, item 1a asserts that if state s' in N has a *must*-transition $L(s', a, \varphi')$, then from state s in P there must be at least one transition $L(s, a, \mu)$ such that the distribution μ is simulated by some distribution $\mu' \in Sat(\varphi')$. Item 1b requires that each transition $L(s, a, \mu)$ from state s in P must have a corresponding *must-may*-transition $L(s', a, \varphi')$ from state s' in N such that the distribution μ is simulated by some distribution $\mu' \in Sat(\varphi')$. Items 2a and 2b are as before.

We write $P \models N$ iff there exists a satisfaction relation between s_0 and s'_0 , and call P an *implementation* of N . The set of all implementations of N is given by $\llbracket N \rrbracket = \{P \mid P \models N\}$.

Example 4.3.2. The PA P depicted in Figure 4.1 is an implementation of the ACPA N in Figure 4.3. Note that P implements all the required behavior of N , and the whole behavior of P is reflected in N . $R = \{(s_0, s_0), (s_1, s_1), (s_2, s_2), (s_3, s_3)\}$ is a satisfaction relation between P and N .

4.3.2 Consistency

In this section we discuss the concept of consistency for ACPA. We start with the definition of consistency for a state and then lift it for ACPA.

Definition 4.3.5. (State Consistency) A state s is *consistent* if it is *valuation consistent* (has at least one *must*\backslash*may*-valuation) as well as *constraint consistent* (for every *must*-transition (s, a, φ) , the satisfaction set $Sat(\varphi)$ is not empty).

Definition 4.3.6. (ACPA Consistency) An ACPA is *consistent* if all of its states are consistent.

A consistent ACPA has at least one implementation. An ACPA is inconsistent if its initial state is inconsistent. Moreover, it is also possible that all states of an ACPA are not consistent; however, this does not imply that the specification given as ACPA is inconsistent. We remove all inconsistent states of an ACPA and as a result we get a new ACPA which may still contain some inconsistent states. This process is repeated until no further removal of inconsistent states is possible. If the resulting ACPA has its initial state consistent, then it is consistent; and it has the same set of models (implementations) as the original ACPA. We call this process a *pruning operation*.

In the following, we give a formal definition of a function that removes only inconsistent states from ACPA, and later we use it in the formal definition of the pruning operation.

Definition 4.3.7. Let $N = (S, A, L, AP, V, s_0)$ be an ACPA with a dummy state $\lambda \notin S$ and a set $T \subseteq S$ of inconsistent states. A function $\nu : S \rightarrow \{\lambda\} \cup S \setminus T$ maps every inconsistent state onto λ such that for all $s \in S$:

$$\nu(s) = \begin{cases} \lambda & \text{if } \forall p \in AP : V(s, p) = \perp & /* \textit{valuation inconsistent} \\ \text{or } \exists a \in A, \exists \varphi \in C(S) : L(s, a, \varphi) = \top \wedge Sat(\varphi) = \emptyset & /* \textit{constraint inconsistent} \\ s & \textit{otherwise} \end{cases}$$

Now we give a formal definition of the pruning operation as:

Definition 4.3.8. (Pruning) Let $N = (S, A, L, AP, V, s_0)$ be an ACPA with $\lambda \notin S$ and a set $T \subseteq S$ of inconsistent states. Let $\nu : S \rightarrow \{\lambda\} \cup S \setminus T$ be a function as defined above. Let β be a pruning function that induces the ACPA $\beta(N) = (S', A, L', AP, V', s'_0)$ such that $S' = \{s' \neq \lambda \mid \exists s \in S : \nu(s) = s'\} \cup \{s'_0\}$, and for all $s' \in S'$ we have:

For each $\varphi' \in C(S')$, there exists $\varphi \in C(S)$, and vice versa, such that:

$$L'(s', a, \varphi') = L(s', a, \varphi)$$

where φ and φ' are related as follows:

$$\begin{aligned} \forall \mu \in \text{Sat}(\varphi) : (\mu(r) = 0 \quad \text{for all } r \in T) \\ \implies \exists \mu' \in \text{Sat}(\varphi') : (\mu'(s') = \mu(s') \quad \text{for all } s' \in S') \\ \iff \\ \forall \mu' \in \text{Sat}(\varphi'), \exists \mu \in \text{Sat}(\varphi) : (\mu'(s') = \mu(s') \quad \text{for all } s' \in S') \\ \implies (\mu(r) = 0 \quad \text{for all } r \in T) \end{aligned}$$

Note that all states in T are mapped to λ and they are removed from ACPA N . The ACPA $\beta(N)$ that we achieve as a result of the *pruning* operation may still contain some inconsistent states. Therefore, we repeat this process until a fixpoint is achieved such that $\beta^n(N) = \beta^{n+1}(N)$, where n represents the number of iterations. Mathematically, the fixpoint of β is given as $\beta^*(N) = \lim_{n \rightarrow \infty} \beta^n(N)$. The existence of this fixpoint is guaranteed as N is finite.

The operations (*conjunction* and *composition*) that we discuss later may introduce inconsistent states in the system. Therefore, we use the *pruning* operation to get rid of such states after these operations.

Lemma 1. *For any ACPA N , $\beta(N)$ is an ACPA.*

Example 4.3.3. Consider the ACPA $N = (S, A, L, AP, V, s_0)$ (left) as depicted in Figure 4.4. As the state s_3 in N does not have any *must-* or *may-*valuation, it is an inconsistent state. Let β be a *pruning* function such that $\beta(N)$ (right) is the induced ACPA. Note that the inconsistent state s_3 in N has been removed in $\beta(N)$, and the labeling of each state in $\beta(N)$ is also preserved. Moreover, note that any distribution μ with $\mu(s_3) = 0$ belongs to $\text{Sat}(\varphi)$ iff it also belongs to $\text{Sat}(\varphi')$. In fact, the constraint function φ' is obtained from φ by setting $x_3 = 0$ and tightening the bounds of the remaining constraints. Note that we do not

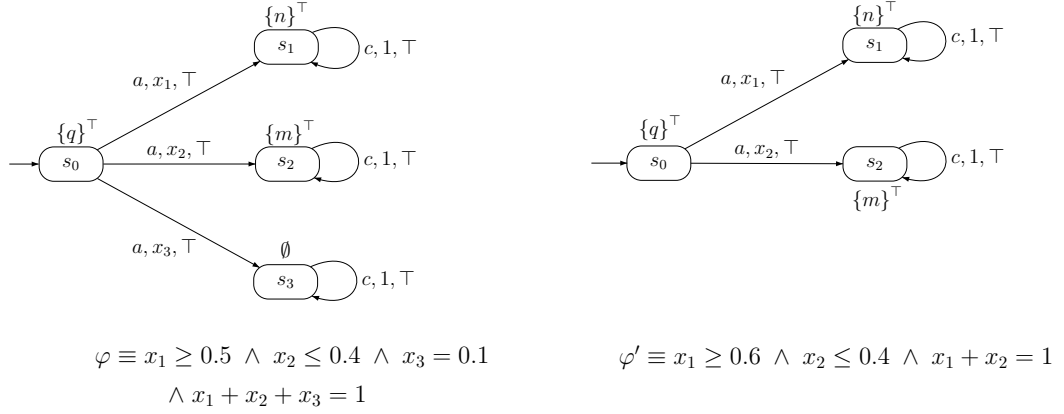


Figure 4.4: An example of a pruning operation.

disturb the bound of the constraint $x_2 \leq 0.4$ because if we have changed it to $x_2 \leq 0.5$, there would be distributions in $Sat(\varphi')$ with $\mu(x_3) = 0$ that are not present in $Sat(\varphi)$. Therefore, we change the bound of the constraint $x_1 \geq 0.5$ to $x_1 \geq 0.6$. As all states in $\beta(N)$ are consistent, thus, it is also consistent and no more *pruning* operation is required.

The following theorem tells that if a PA P satisfies an ACPA N , it also satisfies the induced ACPA $\beta(N)$. Moreover, the *implementation* sets of N and $\beta^*(N)$ are equal.

Theorem 1. *Let N be an ACPA, then for any PA P we have:*

- (1) $P \models N \iff P \models \beta(N)$, and
- (2) $\llbracket N \rrbracket = \llbracket \beta^*(N) \rrbracket$

This is similar to Theorem 2 of [CDL⁺10].

4.3.3 Abstraction

In this section we explain how an ACPA is abstracted by partitioning its state space, i.e, by grouping sets of states into abstract ones. First, we lift the Definition 3.2.3 of an abstraction function α from states to distributions, which is then used in the formal definition of abstraction for ACPA.

Definition 4.3.9. Let $\alpha : S \rightarrow S'$ be an abstraction function as defined in Definition 3.2.3, then for each distribution $\mu \in Dist(S)$, $\alpha(\mu) \in Dist(\alpha(S))$ is given as:

$$\alpha(\mu)(s') = \mu(\alpha^{-1}(s')) \quad \text{for all } s' \in \alpha(S)$$

and for a set of distributions $X \subseteq \text{Dist}(S)$, we have:

$$\alpha(X) = \bigcup_{\mu \in X} \alpha(\mu)$$

From the above definition, it implies that $\varphi' = \alpha(\varphi)$ iff $\text{Sat}(\varphi') = \alpha(\text{Sat}(\varphi))$. The abstraction of the product of two constraint functions φ and φ' is given as $\alpha(\varphi \cdot \varphi') = \alpha(\varphi) \cdot \alpha(\varphi')$. Now we use Definition 3.2.5 of abstraction for an MTS and extend it for an ACPA. Formally, it is given as follows:

Definition 4.3.10. (Abstraction) For an ACPA $N = (S, A, L, AP, V, s_0)$, the abstraction function $\alpha : S \rightarrow S'$ induces the ACPA $\alpha(N) = (S', A, L', AP, V', s'_0)$, where for all $a \in A$, $s', u' \in S'$, and $\varphi' \in C(S')$:

$$L'(s', a, \varphi') = \begin{cases} \top & \text{if } \forall s \in \gamma(s') : \exists \varphi \in C(S) : L(s, a, \varphi) = \top, \text{ and} \\ & \text{Sat}(\varphi') = \alpha\left(\bigcap_{s \in \gamma(s'), \exists \varphi \in C(S) : L(s, a, \varphi) = \top} \text{Sat}(\varphi)\right) \quad (\text{a}) \\ \\ ? & \text{if } \forall s \in \gamma(s') : \exists \varphi \in C(S) : L(s, a, \varphi) = \top, \text{ and} \\ & \text{Sat}(\varphi') = \alpha\left(\bigcup_{s \in \gamma(s'), \exists \varphi \in C(S) : L(s, a, \varphi) = \top} \text{Sat}(\varphi)\right) \setminus \\ & \alpha\left(\bigcap_{s \in \gamma(s'), \exists \varphi \in C(S) : L(s, a, \varphi) = \top} \text{Sat}(\varphi)\right) \quad (\text{b1}) \\ \\ \text{or} \\ & \text{if } \exists s \in \gamma(s') : \exists \varphi \in C(S) : L(s, a, \varphi) \neq \perp, \text{ and} \\ & \text{Sat}(\varphi') = \alpha\left(\bigcup_{s \in \gamma(s'), \exists \varphi \in C(S) : L(s, a, \varphi) \neq \perp} \text{Sat}(\varphi)\right) \quad (\text{b2}) \\ \\ \perp & \text{otherwise} \quad (\text{c}) \end{cases}$$

and for all $p \in AP$,

$$V'(s', p) = \begin{cases} \top & \text{if } \forall s \in \gamma(s') : V(s, p) = \top \\ \perp & \text{if } \forall s \in \gamma(s') : V(s, p) = \perp \\ ? & \text{otherwise} \end{cases}$$

Let us explain this definition briefly. Item (a) asserts that the *must* a -transition (s', a, φ') from an abstract state s' represents the common behavior of the *must* a -transitions (s, a, φ) of all states $s \in \gamma(s')$. Items (b1) and (b2) assert that the *may* a -transition from an abstract state s' represents either the uncommon behavior of the *must* a -transitions (s, a, φ) of all states $s \in \gamma(s')$ or if some states (not all) in $\gamma(s')$ do not have a *must* a -transition, it represents the whole behavior of

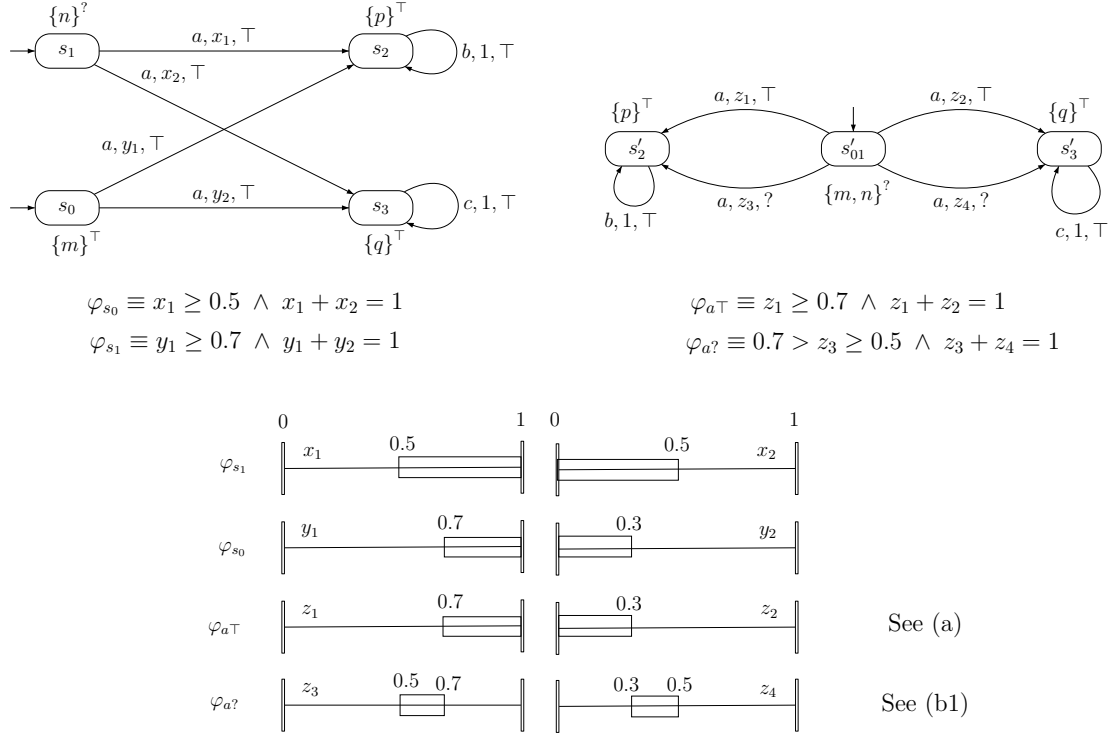


Figure 4.5: The ACPA N (left) is abstracted by the ACPA N' (right), i.e., $N' = \alpha(N)$

a -transitions (s, a, φ) of all states $s \in \gamma(s')$. Item (c) asserts that if non of the states in $\gamma(s')$ have an a -transition, then the abstract state s' also does not have an a -transition.

Lemma 2. *For any ACPA N , $\alpha(N)$ is an ACPA.*

Example 4.3.4. Consider the ACPA $N = (S, A, L, AP, V, s_0)$ (left, top) and $N' = (S', A, L', AP, V', s'_0)$ (right, top) in Figure 4.5. Let the abstraction function $\alpha : S \rightarrow S'$ be given by $\alpha(s_0) = s'_{01} = \alpha(s_1)$, $\alpha(s_2) = s'_2$ and $\alpha(s_3) = s'_3$. Note that, although, both initial states in N have a *must* a -transition, but in abstract state s'_{01} in N' we do not represent both of them by a single *must* a -transition. Rather, we have two a -transitions in s'_{01} : a *must* a -transition and a *may* a -transition. This is because the satisfaction sets of φ_{s_0} and φ_{s_1} are not disjoint. The *must* a -transition $(s'_{01}, a, \varphi_{a\top})$ in N' represents the common behavior of transitions (s_0, a, φ_{s_0}) and (s_1, a, φ_{s_1}) in N ; whereas the *may* a -transition $(s'_{01}, a, \varphi_{a?})$ represents the uncommon behavior of the corresponding transitions. In the bottom of the Figure 4.5, we

show pictorially the constraint functions φ_0 and φ_1 along with their overlapping and differences given by the constraint functions $\varphi_{a\top}$ and $\varphi_{a?}$ respectively.

4.3.4 Refinement

In this section we discuss how specifications of probabilistic systems can be compared to each other. Specifications are usually compared using a *refinement* relation. In the sequel, we discuss a refinement relation for ACPA. We use Definition 3.2.6 of refinement for MTSs and extend it for ACPA.

First we give a definition of a simulation relation between constraint functions, and then use it in the definition of a refinement relation between ACPA.

Definition 4.3.11. Let $N = (S, A, L, AP, V, s_0)$ and $N' = (S', A', L', AP', V', s'_0)$ be ACPA and $R \subseteq S \times S'$. Then for $\varphi \in C(S)$ and $\varphi' \in C(S')$:

$$\begin{aligned} \varphi \sqsubseteq_R \varphi' \\ \iff \\ \forall \mu \in \text{Sat}(\varphi), \exists \mu' \in \text{Sat}(\varphi') : \mu \sqsubseteq_R \mu' \end{aligned}$$

Now we give a formal definition of a refinement relation for ACPA.

Definition 4.3.12. (Refinement) Let $N = (S, A, L, AP, V, s_0)$ and $N' = (S', A', L', AP', V', s'_0)$ be two ACPA with $AP \subseteq AP'$ and $A \subseteq A'$. $R \subseteq S \times S'$ is a refinement relation, iff for all sRs' , the following conditions hold:

- 1a. $\forall a \in A', \forall \varphi' \in C(S') : L'(s', a, \varphi') = \top \implies \exists \varphi \in C(S) : L(s, a, \varphi) = \top \wedge \varphi \sqsubseteq_R \varphi'$
- 1b. $\forall a \in A, \forall \varphi \in C(S) : L(s, a, \varphi) \neq \perp \implies \exists \varphi' \in C(S') : L'(s', a, \varphi') \neq \perp \wedge \varphi \sqsubseteq_R \varphi'$
- 2a. $\forall p \in AP' : V'(s', p) = \top \implies V(s, p) = \top$
- 2b. $\forall p \in AP : V(s, p) \neq \perp \implies V'(s', p) \neq \perp$

Let us briefly explain this definition. Item 1a requires that a *must*-transition $L(s', a, \varphi')$ from state s' in N' must have a corresponding *must*-transition $L(s, a, \varphi)$ from state s in N such that the constraint function φ is simulated by the constraint function φ' . Item 1b requires that each transition $L(s, a, \varphi)$ from state s must have

a corresponding *must-may*-transition $L(s', a, \varphi')$ from state s' such that the constraint function φ is simulated by the constraint function φ' . Items 2a and 2b are as before.

Remark. A satisfaction relation is a special type of refinement relation.

We write $s \preceq s'$ iff sRs' for some refinement relation R , and $N \preceq N'$, for ACPA N and N' with initial states s_0 and s'_0 , iff $s_0 \preceq s'_0$. The following theorem tells that there exists a refinement relation between an ACPA and its abstraction.

Theorem 2. *For any ACPA N and abstraction function α :*

$$N \preceq \alpha(N)$$

Example 4.3.5. Consider the ACPA N and N' given in Example 4.3.4. As N' is an abstraction of N , it follows $N \preceq N'$.

The following theorem tells that if an ACPA N is a refinement of an ACPA N' , the implementation set of N is a subset of that of N' .

Theorem 3. *For ACPA N and N' :*

$$N \preceq N' \implies \llbracket N \rrbracket \subseteq \llbracket N' \rrbracket$$

4.3.5 Conjunction

The *conjunction* operation is an important part of every specification theory. It is used to take the intersection of the requirements expressed by two or more specifications. The result of conjunction of two different ACPA is an ACPA. We use Definition 3.2.7 of conjunction for MTSs and extend it for ACPA. Formally, it is given as:

Definition 4.3.13. (Conjunction) Let $N = (S, A, L, AP, V, s_0)$ and $N' = (S', A', L', AP', V', s'_0)$ be ACPA. The conjunction of N and N' , written as $N \wedge N'$, is given as:

$$N \wedge N' = (S \times S', A \cap A', \tilde{L}, AP \cap AP', \tilde{V}, (s_0, s'_0))$$

where for all $(s, s') \in S \times S'$, $a \in A \cap A'$ and $\tilde{\varphi} \in C(S \times S')$, there exists $\varphi \in C(S)$ and $\varphi' \in C(S')$:

$$\tilde{L}((s, s'), a, \tilde{\varphi}) = L(s, a, \varphi) \oplus L'(s', a, \varphi')$$

4.3. ABSTRACT CONSTRAINT PROBABILISTIC AUTOMATA (ACPA)

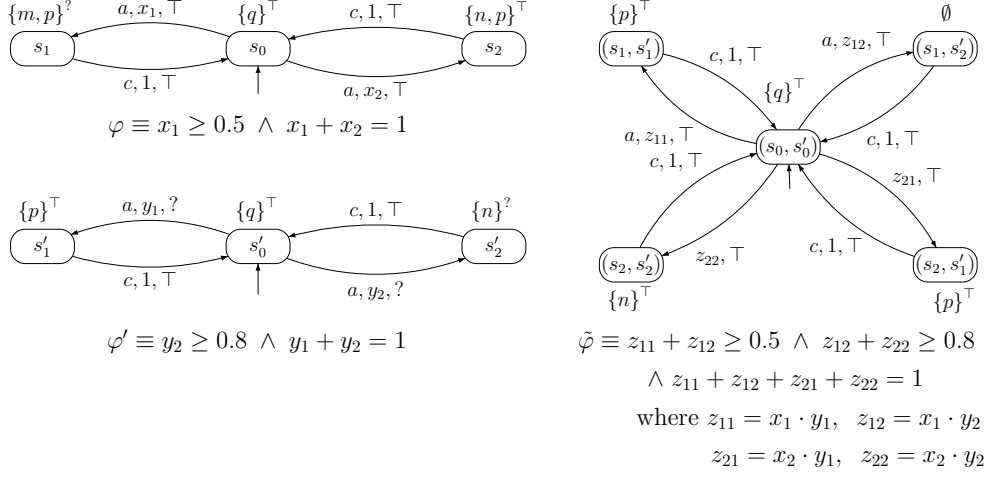


Figure 4.6: The ACPA N (left top) and N' (left bottom) and their conjunction $N \wedge N'$ (right).

If $\tilde{L}((s, s'), a, \tilde{\varphi}) \neq \perp$, then $\tilde{\varphi} = \varphi \cdot \varphi'$ and for all $u \in S, u' \in S'$ and $\tilde{\mu} \in \text{Sat}(\tilde{\varphi})$:

$$\begin{aligned} \tilde{\mu}((u, S')) &= \mu(u) && \text{for some } \mu \in \text{Sat}(\varphi) \\ \tilde{\mu}((S, u')) &= \mu'(u') && \text{for some } \mu' \in \text{Sat}(\varphi') \end{aligned}$$

and for all $p \in AP \cap AP'$,

$$\tilde{V}((s, s'), p) = V(s, p) \oplus V'(s', p)$$

Conjunction may result in some inconsistent states in a system, therefore, it must be followed by pruning. The following theorem tells that the conjunction of two ACPA preserves refinement relations.

Theorem 4. *Let M, N, O and P be ACPA, then:*

$$(M \preceq N) \wedge (O \preceq P) \implies (M \wedge O) \preceq (N \wedge P)$$

As for any ACPA N , $N \wedge N = N$. We have the following corollary:

Corollary. *Let M, N and O be ACPA, then:*

$$(M \preceq N) \wedge (M \preceq O) \implies M \preceq (N \wedge O)$$

Example 4.3.6. Consider the ACPA $N = (S, A, L, AP, V, s_0)$ (left top) and $N' = (S', A', L', AP', V', s'_0)$ (left bottom) and their conjunction $N \wedge N' = (S \times S', A \cap A', \tilde{L}, AP \cup AP', \tilde{V}, (s_0, s'_0))$ (right) depicted in Figure 4.6. Both N and N' have an a -transition from their initial states s_0 and s'_0 but s_0 has a *must* a -transition (s_0, a, φ) whereas s'_0 has a *may* a -transition (s'_0, a, φ') ; therefore, the composite state (s_0, s'_0) has a *must* a -transition $((s_0, s'_0), a, \tilde{\varphi})$. Note that for each $\tilde{\mu} \in \text{Sat}(\tilde{\varphi})$, there is $\mu \in \text{Sat}(\varphi)$ and $\mu' \in \text{Sat}(\varphi')$ such that $\tilde{\mu}((u, S')) = \mu(u)$ and $\tilde{\mu}((S, u')) = \mu'$, where $u \in S$ and $u' \in S'$. Moreover, the valuations sets on state (s_1, s'_2) are empty. This is because neither s_1 nor s'_2 has any valuation in common in N and N' . Hence, the state (s_1, s'_2) is inconsistent and it would be removed if a pruning operation is applied.

Remark. For ACPA N and N' , $N \wedge N' \preceq N$ may not hold. This is because if some *must* behavior of N is not in common with N' , it would also be absent in $N \wedge N'$; hence, it would violate the condition 1a of Definition 4.3.12. However, in case of CMCs [CDL⁺10], such proposition always holds.

4.3.6 Parallel Composition

Parallel composition is an operation in which several automata are combined into a single composite automaton that allows the interleaving of the actions of the component automata. This operation helps in compositional modeling of complex systems. Each component is modeled by an independent automaton and these automata are then combined into a single composite automaton by parallel composition. In composite automaton each transition is either *synchronous* or *asynchronous*. In synchronous case all participating automata are required to perform their transitions simultaneously on some common action. Whereas in asynchronous case one of the participating automata performs a transition independently of others. We use Definition 3.2.8 of parallel composition for MTSs and extend it for ACPA. Formally it is given as:

Definition 4.3.14. (Parallel Composition) Let $N = (S, A, L, AP, V, s_0)$ and $N' = (S', A', L', AP', V', s'_0)$ be ACPA with $AP \cap AP' = \emptyset$. The parallel composition of N and N' w.r.t. *synchronization set* $\bar{A} \subseteq A \cap A'$ is given as:

$$N \parallel_{\bar{A}} N' = (S \times S', A \cup A', \tilde{L}, AP \cup AP', \tilde{V}, (s_0, s'_0))$$

where for all $(s, s') \in S \times S'$, $a \in A \cup A'$ and $\tilde{\varphi} \in C(S \times S')$, there exists $\varphi \in C(S)$ and $\varphi' \in C(S')$:

$$\tilde{L}((s, s'), a, \tilde{\varphi}) = \begin{cases} L(s, a, \varphi) & \text{if } a \in A \setminus \bar{A} \\ L'(s', a, \varphi') & \text{if } a \in A' \setminus \bar{A} \\ L(s, a, \varphi) \sqcap L'(s', a, \varphi') & \text{if } a \in \bar{A} \end{cases}$$

If $\tilde{L}((s, s'), a, \tilde{\varphi}) \neq \perp$, then for all $u \in S$, $u' \in S'$ and $\tilde{\mu} \in \text{Sat}(\tilde{\varphi})$:

$$\tilde{\mu}((u, u')) = \begin{cases} \mu(u) \cdot I(s', u') & a \in A \setminus \bar{A}, \mu \in \text{Sat}(\varphi) \\ \mu'(u') \cdot I(s, u) & a \in A' \setminus \bar{A}, \mu' \in \text{Sat}(\varphi') \\ \mu(u) \cdot \mu'(u') & a \in \bar{A}, \mu \in \text{Sat}(\varphi), \mu' \in \text{Sat}(\varphi') \end{cases}$$

and for all $p \in AP \cup AP'$,

$$\tilde{V}((s, s'), p) = \begin{cases} V(s, p) & \text{if } p \in AP \\ V'(s', p) & \text{if } p \in AP' \end{cases}$$

For simplicity we assume that the set of valuations AP and AP' are disjoint; otherwise, we may exploit *three*-valued propositional labeling.

Example 4.3.7. Consider the ACPA $N = (S, A, L, AP, V, s_0)$ (left top) and $N' = (S', A', L', AP', V', s'_0)$ (left bottom) and their parallel composition $N \parallel_{\bar{A}} N' = (S \times S', A \cup A', \tilde{L}, AP \cup AP', \tilde{V}, (s_0, s'_0))$ (right) w.r.t. synchronization actions $\{a\} \subseteq A \cap A'$ depicted in Figure 4.7. Both N and N' synchronize on action a from their initial states s_0 and s'_0 but s_0 has a *must* a -transition (s_0, a, φ_a) whereas s'_0 has a *may* a -transition (s'_0, a, φ'_a) ; therefore, the composite state (s_0, s'_0) has a *may* a -transition $((s_0, s'_0), a, \tilde{\varphi}_a)$. Also note that $\text{Sat}(\tilde{\varphi}_a) = \text{Sat}(\varphi_a) \cdot \text{Sat}(\varphi'_a)$. Moreover, N' has a *must* b -transition (s'_0, b, φ'_b) from its initial state, that is depicted as a *must* b -transition $((s_0, s'_0), b, \tilde{\varphi}_b)$ from the composite state (s_0, s'_0) such that $\text{Sat}(\tilde{\varphi}_b) = \text{Sat}(\varphi'_b \cdot I(s_0, \cdot))$. Furthermore, note that (s_1, s'_1) has two set of valuations: $\{o\}$ is a *must*-valuations set and $\{m\}$ is a *may*-valuations set, and they satisfy valuations of both constituent states s_1 and s'_1 .

In the following definition we explain that the product of two constraint functions preserves simulation relations. Later on, we will use this definition in the proof of Theorem 5.

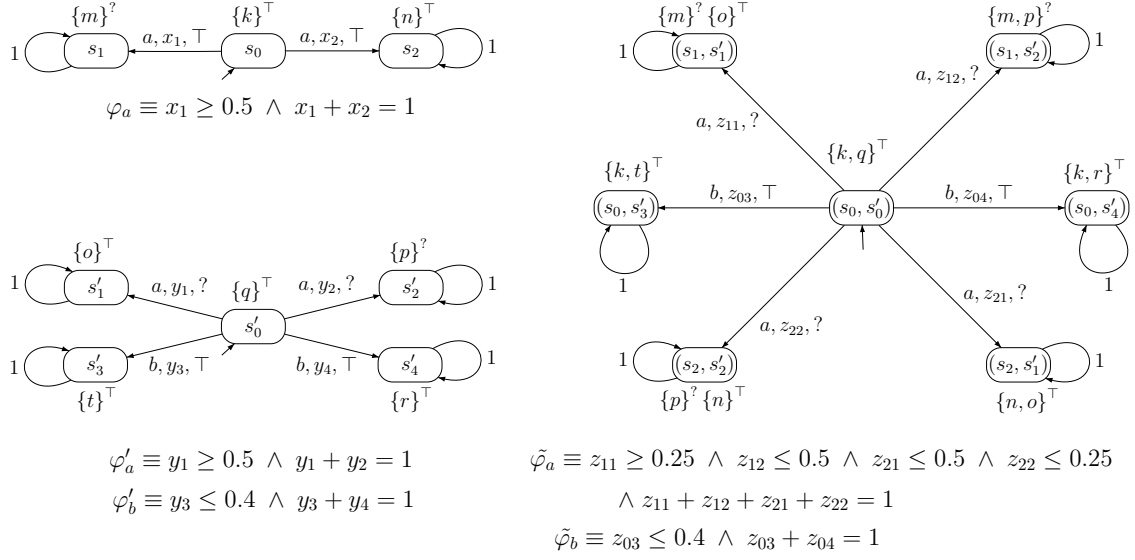


Figure 4.7: The ACPA N (left top) and N' (left bottom) and their parallel composition $N \parallel_{\bar{A}} N'$ w.r.t. synchronization actions $\bar{A} = \{a\}$.

Definition 4.3.15. Let $M = (S, A, L, AP, V, s_0)$, $N = (S', A', L', AP', V', s'_0)$, $O = (S'', A'', L'', AP'', V'', s''_0)$ and $P = (S''', A''', L''', AP''', V''', s'''_0)$ be ACPA. Let $R \subseteq S \times S'$ and $R' \subseteq S'' \times S'''$ be simulation relations such that $\tilde{R} = \{((s, s''), (s', s''')) \mid sRs' \wedge s''R's'''\}$, then for $\varphi \in C(S)$, $\varphi' \in C(S')$, $\varphi'' \in C(S'')$ and $\varphi''' \in C(S''')$:

$$\varphi \cdot \varphi'' \sqsubseteq_{\tilde{R}} \varphi' \cdot \varphi'''$$

$$\iff$$

$$\forall \mu \in \text{Sat}(\varphi), \forall \mu'' \in \text{Sat}(\varphi''), \exists \mu' \in \text{Sat}(\varphi'), \exists \mu''' \in \text{Sat}(\varphi''') : \mu \cdot \mu'' \sqsubseteq_{\tilde{R}} \mu' \cdot \mu'''$$

The following theorem explains that refinement is a precongurence with respect to parallel composition, thus, enabling compositional abstraction of ACPA.

Theorem 5. *Refinement \preceq is a precongurence w.r.t. $\parallel_{\bar{A}}$.*

Corollary. *For PA P and P' , and ACPA N and N' :*

$$P \models N \wedge P' \models N' \implies P \parallel_{\bar{A}} P' \models N \parallel_{\bar{A}} N'$$

The following theorem proposes that if we first apply abstraction on components of a system separately and then compose them into a composite model, in

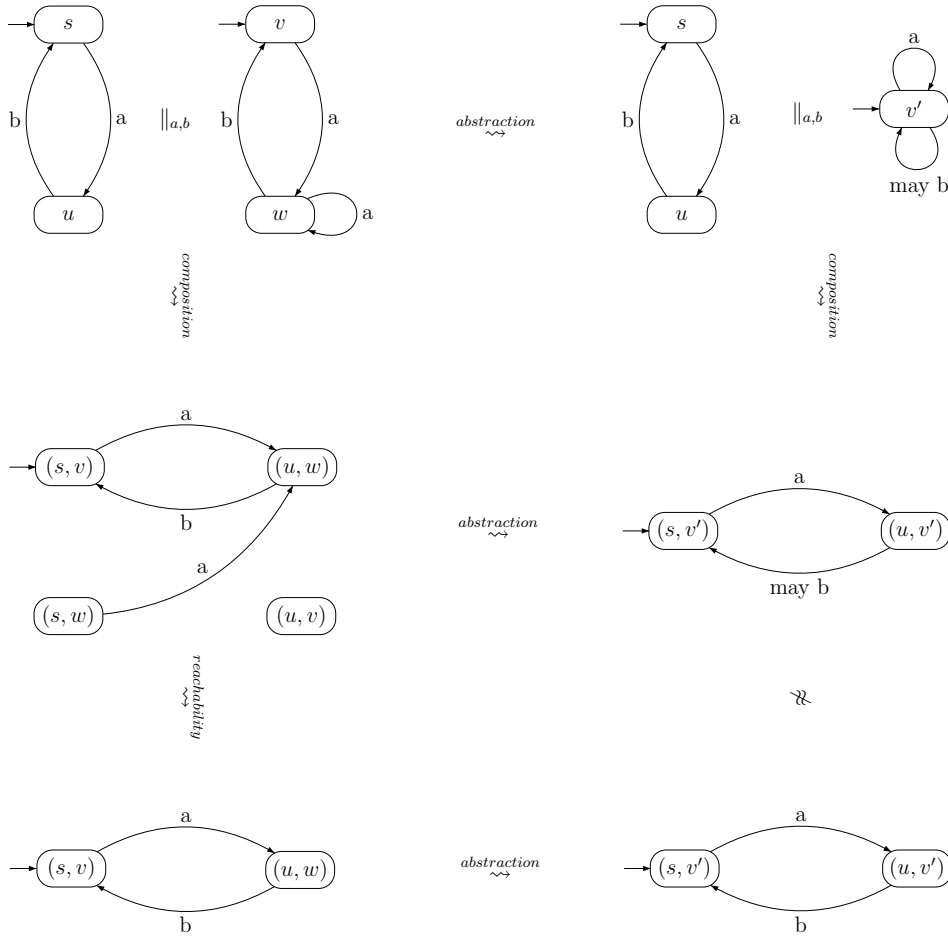


Figure 4.8: Performing abstraction on component level vs. on monolithic system

fact, we get the same composite model as by first composing the components into a monolithic model and then abstracting it. However, in the later case after composing the components without prior abstraction, some unreachable states may appear in the monolithic model which are normally removed by a reachability analysis. And if we abstract the monolithic model after removing unreachable states, we do not get the same composite model as we get in the former case.

Theorem 6. *Let M and N be ACPA, \bar{A} a synchronization set, and α_1 , α_2 and α_3 be abstraction functions such that $\alpha_3 = \alpha_1 \times \alpha_2$, then:*

$$\alpha_1(M) \parallel_{\bar{A}} \alpha_2(N) = \alpha_3(M \parallel_{\bar{A}} N) \quad \text{up to isomorphism}$$

We consider an example of an MTS: an APA in which for every distribution μ , the support $\text{supp}(\mu)$ is singleton.

Example 4.3.8. Consider the MTSs in Figure 4.8 (top, left). When these MTSs are composed we get the monolithic model (middle, left). Now when we apply abstraction on this model, we get the MTS (middle, right); this is the same MTS that we get by first applying abstraction on MTSs (top, left) individually and then composing them (top, right) into composite MTS (middle, right). However, the monolithic model (middle, left) have unreachable states (s, w) and (u, v) . When these states are removed by reachability analysis, we get another MTS (bottom, left), which if abstracted (bottom, right) does not coincide with the MTS (middle, right).

4.3.7 Valuation Synchronization

Valuation synchronization is a process by which we make sure that the *must*-valuations of every state in ACPA are admissible, i.e., they do not violate the conditions given in the requirements of a system. There are two ways by which *must*-valuations in a state may become inadmissible: parallel composition and refinement. In parallel composition the *must*-valuations of a composite state is a union of *must*-valuations of constituent states, so, if they are inadmissible mutually, the *must*-valuations in the composite state are also inadmissible. Similarly, the *must*- and *may*-valuations of a state in ACPA may be inadmissible mutually. Due to this, all possible refinements of a state may not have admissible *must*-valuations. For example, a state s has $\{a, b\}^\top$ and $\{c, d\}^\top$ as *must*- and *may*-valuations sets; then, the *must*-valuations sets that can appear in any refinement of S are $\{a, b\}$, $\{a, b, c\}$, $\{a, b, d\}$ and $\{a, b, c, d\}$. Suppose a and d are in conflict, i.e., they cannot be together in any *must*-valuations set. Then, any refinement of s that has either $\{a, b, d\}$ or $\{a, b, c, d\}$ as *must*-valuations is *valuation inconsistent*. In short, after composition or refinement it is necessary to make sure that the *must*-valuations of each state are admissible. This is done by a *valuation synchronization* operation. Formally, we define it as follows:

Definition 4.3.16. Let $N = (S, A, L, AP, V, s_0)$ be an ACPA. Let $\omega : S \rightarrow S$ be a synchronization function that detects inadmissible *must*-valuations in states such that $\omega(N) = (S, A, L, AP, V', s_0)$ is the induced ACPA. Let K be a set of all admissible valuations over AP , then for all $s \in S$ and $p \in AP$:

$$V'(s, p) = \begin{cases} V(s, p) & \text{if } \{q \mid \forall q \in AP : V(s, q) = \top\} \in K \\ \perp & \text{otherwise} \end{cases}$$

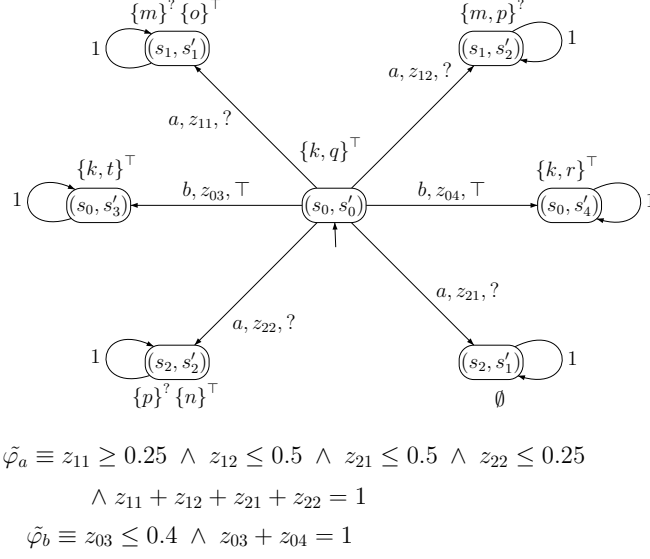


Figure 4.9: The ACPA given in this figure is obtained by valuation synchronization operation on the ACPA given in Figure 4.7 (right).

Example 4.3.9. Consider the ACPA $N = (S, A, L, AP, V, s_0)$ (left top) and $N' = (S', A', L', AP', V', s'_0)$ (left bottom) and their parallel composition $N \parallel_{\bar{A}} N' = (S \times S', A \cup A', \tilde{L}, AP \cup AP', \tilde{V}, (s_0, s'_0))$ (right) w.r.t. synchronization actions $\{a\} \subseteq A \cap A'$ depicted in Figure 4.7. Let $n \in AP$ and $o \in AP'$ cannot be *must*-valuations together in any composite state. Let the set of admissible valuations over AP be given as $K = \{\{m, o\}, \{m, p\}, \{k, r\}, \{k, q\}, \{k, t\}, \{p, n\}\}$. Let $\omega : S \rightarrow S$ be a synchronization function that detects inadmissible *must*-valuations in states such that the induced ACPA $\omega(N \parallel_{\bar{A}} N')$ is given in Figure 4.9. Note that in $N \parallel_{\bar{A}} N'$ the set of *must*-valuations of the state (s_2, s'_1) is $\{n, o\}$ that does not belong to K ; therefore, this state has become valuation inconsistent in $\omega(N \parallel_{\bar{A}} N')$ and, hence, can be removed by pruning.

Through out this work we have considered atomic propositions. However, to define a probabilistic logic for expressing properties of ACPA is out of the scope of this thesis.

5

Conclusion

In this thesis we lay down the foundations of a compositional design methodology for probabilistic systems. We propose a new abstract model, abstract constraint probabilistic automata (ACPA), and use it to construct a specification theory for probabilistic automata (PA). We start our work with an overview of specification theory of modal transition systems (MTSs) that is used in compositional design of non-probabilistic systems. We discuss the notions of satisfaction, abstraction, refinement as well as conjunction and parallel composition for modal transition systems (MTSs).

In our main work, we equip probabilistic automata (PA) with the features of modal transition systems (MTSs) and constraint Markov chains (CMCs). We generalize PA to ACPA by using the notions of *must* and *may*-transitions as well as *must* and *may*-valuations as introduced for MTSs, and by specifying the target of every transition by a (possibly infinite) set of distributions as in CMCs. Thus, ACPA have all the features of MTSs as well as CMCs. Our specification theory comes with the constructs for abstraction, refinement, consistency checking, parallel composition, conjunction as well as valuation synchronization - all indispensable ingredients for a compositional design methodology. Moreover, our work on compositional abstraction for ACPA is a step further in this area of compositional modeling. We propose to perform abstraction on component level which can then be composed into monolithic model for the purpose of analysis and verification. This helps in reducing peak memory requirements during the construction of a monolithic model of a system.

Moreover, our abstraction, ACPA, is definitely an extension of CMC presented in [CDL⁺10]. This is because every CMC is an ACPA in which only a single constraint function is possible in each state.

In the future, it would be of interest to define a quotient operation for ACPA, thus, completing the specification theory for PA. It would also be interesting to

design, implement and evaluate efficient algorithms for abstracting an ACPA, finding whether an ACPA refines another ACPA, checking consistency of ACPA, and for conjoining and composing ACPA. This work can further be extended by partitioning actions of ACPA into *internal* and *external* ones as in [KKN09]. Moreover, one can develop a probabilistic logic for expressing properties of systems modeled by ACPA. Furthermore, one can provide an industrial case-study based on this work; and can investigate the applicability of our approach in model checking procedures.

A

Proofs of Theorems

Lemmas

Lemma 3. *Let $\mu \in \text{Dist}(S)$, $\mu' \in \text{Dist}(S')$ and $\mu'' \in \text{Dist}(S'')$. Let $R \subseteq S \times S'$ and $R' \subseteq S' \times S''$ be simulation relations such that $R'' = R \circ R'$, then:*

$$\mu \sqsubseteq_R \mu' \wedge \mu' \sqsubseteq_{R'} \mu'' \implies \mu \sqsubseteq_{R''} \mu''$$

Proof. In order to prove that $\mu \sqsubseteq_{R''} \mu''$, we need to show that there exists a weight function for distributions μ and μ'' w.r.t. relation R'' . Let Δ and Δ' be the weight functions for distributions μ and μ' w.r.t. relation R , and μ' and μ'' w.r.t. relation R' respectively. We define a weight function for distributions μ and μ'' w.r.t. relation R'' such that for all $s \in S$, $s' \in S'$ and $s'' \in S''$:

$$\Delta''(s, s'') = \sum_{s' \in S'} \frac{\Delta(s, s') \cdot \Delta'(s', s'')}{\mu'(s')}$$

and prove that it fulfills the three properties of a weight function.

1. It follows trivially from the definition.
2. The proof of $\Delta''(s, S'') = \mu(s)$ goes as:

$$\begin{aligned} \sum_{s'' \in S''} \Delta''(s, s'') &= \sum_{s'' \in S''} \sum_{s' \in S'} \frac{\Delta(s, s') \cdot \Delta'(s', s'')}{\mu'(s')} \\ &= \sum_{s' \in S'} \frac{\Delta(s, s') \cdot \sum_{s'' \in S''} \Delta'(s', s'')}{\mu'(s')} \\ &= \sum_{s' \in S'} \frac{\Delta(s, s') \cdot \Delta'(s', S'')}{\mu'(s')} \\ &= \sum_{s' \in S'} \frac{\Delta(s, s') \cdot \mu'(s')}{\mu'(s')} \\ &= \mu(s) \end{aligned}$$

3. It is same as case two. □

Lemma 4. *Let $\mu \in \text{Dist}(S)$, $\mu' \in \text{Dist}(S')$, $\mu'' \in \text{Dist}(S'')$ and $\mu''' \in \text{Dist}(S''')$. Let $R \subseteq S \times S'$ and $R' \subseteq S'' \times S'''$ be simulation relations such that $\tilde{R} = \{((s, s''), (s', s''')) \mid sRs' \wedge s''R's'''\}$, then:*

$$\mu \sqsubseteq_R \mu' \wedge \mu'' \sqsubseteq_{R'} \mu''' \implies \mu \cdot \mu'' \sqsubseteq_{\tilde{R}} \mu' \cdot \mu'''$$

Proof. In order to prove that $\mu \cdot \mu'' \sqsubseteq_{\tilde{R}} \mu' \cdot \mu'''$, we need to show that there exists a weight function for distributions $\mu \cdot \mu''$ and $\mu' \cdot \mu'''$ w.r.t. relation \tilde{R} . Let Δ and Δ' be the weight functions for distributions μ and μ' w.r.t. relation R , and μ'' and μ''' w.r.t. relation R' respectively. We define a weight function for distributions $\mu \cdot \mu''$ and $\mu' \cdot \mu'''$ w.r.t. relation \tilde{R} such that for all $s \in S$, $s' \in S'$, $s'' \in S''$ and $s''' \in S'''$:

$$\Delta''((s, s''), (s', s''')) = \Delta(s, s') \cdot \Delta'(s'', s''')$$

and prove that it fulfills the three properties of a weight function.

1. It follows trivially from the definition.

2. The proof of $\Delta''((s, s''), S' \times S''') = \mu(s) \cdot \mu''(s'')$ goes as

$$\begin{aligned} & \sum_{(s', s''') \in S' \times S'''} \Delta''((s, s''), (s', s''')) \\ &= \sum_{s' \in S'} \sum_{s''' \in S'''} \Delta(s, s') \cdot \Delta'(s'', s''') \\ &= \Delta(s, S') \cdot \Delta'(s'', S''') \\ &= \mu(s) \cdot \mu''(s'') \end{aligned}$$

3. It is same as case two. □

Lemma 5. *Let $\mu \in \text{Dist}(S)$ and $\mu' \in \text{Dist}(S')$. Let $R \subseteq S \times S'$ and $R' \subseteq S'' \times S'''$ be simulation relations such that $\tilde{R} = \{((s, s''), (s', s''')) \mid sRs' \wedge s''R's'''\}$, then for all $(s'', s''') \in R'$:*

$$\mu \sqsubseteq_R \mu' \implies \mu \cdot I(s'', \cdot) \sqsubseteq_{\tilde{R}} \mu' \cdot I(s''', \cdot)$$

Proof. In order to prove that $\mu \cdot I(s'', \cdot) \sqsubseteq_{\tilde{R}} \mu' \cdot I(s''', \cdot)$, we need to show that there exists a weight function for distributions $\mu \cdot I(s'', \cdot)$ and $\mu' \cdot I(s''', \cdot)$ w.r.t. relation \tilde{R} . Let Δ be a weight function for distribution μ and μ' w.r.t. relation R . We define a weight function for distributions $\mu \cdot I(s'', \cdot)$ and $\mu' \cdot I(s''', \cdot)$ w.r.t. relation \tilde{R} such that for all $u \in S$, $u' \in S'$, $u'' \in S''$ and $u''' \in S'''$:

$$\Delta''((u, u''), (u', u''')) = \Delta(u, u') \cdot I(s'', u'') \cdot I(s''', u''')$$

and prove that it fulfills the three properties of a weight function.

1. It follows trivially from the definition.

2. The proof of $\Delta''((u, u''), S' \times S''') = \mu(s) \cdot I(s'', u'')$ goes as

$$\begin{aligned}
& \sum_{(u', u''') \in S' \times S'''} \Delta''((u, u''), (u', u''')) \\
&= \sum_{u' \in S'} \sum_{u''' \in S'''} \Delta(u, u') \cdot I(s'', u'') \cdot I(s''', u''') \\
&= \Delta(u, S') \cdot I(s'', u'') \\
&= \mu(u) \cdot I(s'', u'')
\end{aligned}$$

3. It is same as case two. □

Lemma 6. Let $M = (S, A, L, AP, V, s_0)$, $N = (S', A', L', AP', V', s'_0)$ and $O = (S'', A'', L'', AP'', V'', s''_0)$ be ACPA. Let $\varphi \in C(S)$, $\varphi' \in C(S')$ and $\varphi'' \in C(S'')$. Let $R \subseteq S \times S'$ and $R' \subseteq S' \times S''$ be simulation relations such that $R'' = R \circ R'$, then:

$$\varphi \sqsubseteq_R \varphi' \wedge \varphi' \sqsubseteq_{R'} \varphi'' \implies \varphi \sqsubseteq_{R''} \varphi''$$

Proof. Using Definition 4.3.11, the proof of Lemma 6 goes on the same lines as that of Lemma 3 □

Lemma 7. Let $M = (S, A, L, AP, V, s_0)$, $N = (S', A', L', AP', V', s'_0)$, $O = (S'', A'', L'', AP'', V'', s''_0)$ and $P = (S''', A''', L''', AP''', V''', s'''_0)$ be ACPA. Let $\varphi \in C(S)$, $\varphi' \in C(S')$, $\varphi'' \in C(S'')$ and $\varphi''' \in C(S''')$. Let $R \subseteq S \times S'$ and $R' \subseteq S'' \times S'''$ be simulation relations such that $\tilde{R} = \{(s, s''), (s', s''') \mid sRs' \wedge s''R's'''\}$, then:

$$\varphi \sqsubseteq_R \varphi' \wedge \varphi'' \sqsubseteq_{R'} \varphi''' \implies \varphi \cdot \varphi'' \sqsubseteq_{\tilde{R}} \varphi' \cdot \varphi'''$$

Proof. Using Definition 4.3.15, the proof of Lemma 7 goes on the same lines as that of Lemma 4 □

Lemma 8. Let $M = (S, A, L, AP, V, s_0)$, $N = (S', A', L', AP', V', s'_0)$, $O = (S'', A'', L'', AP'', V'', s''_0)$ and $P = (S''', A''', L''', AP''', V''', s'''_0)$ be ACPA. Let $\varphi \in C(S)$ and $\varphi' \in C(S')$. Let $R \subseteq S \times S'$ and $R' \subseteq S'' \times S'''$ be simulation relations such that $\tilde{R} = \{(s, s''), (s', s''') \mid sRs' \wedge s''R's'''\}$, then for all $(s'', s''') \in R'$:

$$\varphi \sqsubseteq_R \varphi' \implies \varphi \cdot I(s'', \cdot) \sqsubseteq_{\tilde{R}} \varphi' \cdot I(s''', \cdot)$$

Proof. We need to prove that $\forall \mu \in \text{Sat}(\varphi)$, $\exists \mu' \in \text{Sat}(\varphi')$ such that $\mu \cdot I(s'', \cdot) \sqsubseteq_{\tilde{R}} \mu' \cdot I(s''', \cdot)$. Using Definition 4.3.11 and Lemma 5 we can prove this lemma. □

Lemma 9. *Let $\mu \in \text{Dist}(S)$, $\mu' \in \text{Dist}(S')$ and $\mu'' \in \text{Dist}(S'')$. Let $R \subseteq S \times S'$ and $R' \subseteq S \times S''$ be simulation relations such that $\tilde{R} = \{(s, (s', s'')) \mid sRs' \wedge sR's''\}$, then:*

$$\mu \sqsubseteq_R \mu' \wedge \mu \sqsubseteq_{R'} \mu'' \implies \mu \sqsubseteq_{\tilde{R}} \mu' \cdot \mu''$$

Proof. In order to prove that $\mu \sqsubseteq_{\tilde{R}} \mu' \cdot \mu''$, we need to show that there exists a weight function for distributions μ and $\mu' \cdot \mu''$ w.r.t. relation \tilde{R} . Let Δ and Δ' be the weight functions for distributions μ and μ' w.r.t. relation R , and μ and μ'' w.r.t. relation R' respectively. We define a weight function for distributions μ and $\mu' \cdot \mu''$ w.r.t. relation \tilde{R} such that for all $s \in S$, $s' \in S'$ and $s'' \in S''$:

$$\Delta''(s, (s', s'')) = \frac{\Delta(s, s') \cdot \Delta'(s, s'')}{\mu(s)}$$

and prove that it fulfills the three properties of a weight function.

1. It follows trivially from the definition.
2. The proof of $\Delta''(s, S' \times S'') = \mu(s)$ goes as:

$$\begin{aligned} & \sum_{(s', s'') \in S' \times S''} \Delta''(s, (s', s'')) \\ &= \sum_{s' \in S'} \sum_{s'' \in S''} \frac{\Delta(s, s') \cdot \Delta'(s, s'')}{\mu(s)} \\ &= \frac{\Delta(s, S') \cdot \Delta'(s, S'')}{\mu(s)} \\ &= \frac{\mu(s) \cdot \mu(s)}{\mu(s)} \\ &= \mu(s) \end{aligned}$$

3. The proof of $\Delta''(S, (s', s'')) = \mu'(s') \cdot \mu''(s'')$ goes as:

$$\begin{aligned} & \sum_{s \in S} \Delta''(s, (s', s'')) \\ &= \sum_{s \in S} \frac{\Delta(s, s') \cdot \Delta'(s, s'')}{\mu(s)} \\ &= \frac{\sum_{s \in S} \Delta(s, s') \cdot \sum_{s \in S} \Delta'(s, s'')}{\sum_{s \in S} \mu(s)} \\ &= \Delta(S, s') \cdot \Delta'(S, s'') \\ &= \mu'(s') \cdot \mu''(s'') \end{aligned}$$

□

Lemma 10. Let $M = (S, A, L, AP, V, s_0)$, $N = (S', A', L', AP', V', s'_0)$ and $O = (S'', A'', L'', AP'', V'', s''_0)$ be ACPA. Let $\varphi \in C(S)$, $\varphi' \in C(S')$ and $\varphi'' \in C(S'')$. Let $R \subseteq S \times S'$ and $R' \subseteq S \times S''$ be simulation relations such that $\tilde{R} = \{(s, (s', s'')) \mid sRs' \wedge sR's''\}$, then:

$$\varphi \sqsubseteq_R \varphi' \wedge \varphi \sqsubseteq_{R'} \varphi'' \implies \varphi \sqsubseteq_{\tilde{R}} \varphi' \cdot \varphi''$$

Proof. Using Definition 4.3.11, the proof of Lemma 10 goes on the same lines as that of Lemma 9 □

Proof of Theorem 1

Theorem. Let N be an ACPA, then for any PA P we have:

- (1) $P \models N \iff P \models \beta(N)$, and
- (2) $\llbracket N \rrbracket = \llbracket \beta^*(N) \rrbracket$

Proof. Let $P = (S, A, L, AP, V, s_0)$ be a PA and let $N = (S', A', L', AP', V', s'_0)$ be an ACPA such that $T \subseteq S'$ is a set of inconsistent states in N . Let $\nu : S' \rightarrow \{\lambda\} \cup S' \setminus T$ be a function that removes inconsistent states from N . Let $\beta(N) = (S'', A', L'', AP', V'', s''_0)$ such that $S'' = \{s'' \neq \lambda \mid \exists s' \in S' : \nu(s') = s''\} \cup \{s''_0\}$. We first prove that $P \models N \iff P \models \beta(N)$.

“ \implies ” Suppose that $P \models N$. Then there exists a satisfaction relation $R : S \times S'$ such that $s_0Rs'_0$. We define a relation $R' : S \times S''$ as:

$$R' = \{(s, \nu(s')) \mid s' \in R(s)\}$$

and show that R' fulfills (1a),(1b),(2a) and (2b) of Definition 4.3.4.

For each $(s, s'') \in R'$, we have $(s, s') \in R$ and $s'' = \nu(s')$. Then according to Definition 4.3.8:

$$V'(s', p) = V''(s'', p) \quad \text{for all } p \in AP'$$

This shows that R' satisfies conditions (2a) and (2b) of Definition 4.3.4.

Now we prove that (1a) and (1b) of Definition 4.3.4 also hold for R' . We show the proof for item (1b). Item (1a) can be proved in the same way as (1b).

1b. For each $(s, s'') \in R'$ there exists $(s, s') \in R$ such that for all $a \in A'$, according to Definition 4.3.4:

$$\begin{aligned} \forall \mu \in Dist(S), \exists \varphi' \in C(S') : L(s, a, \mu) = \top \\ \implies L'(s', a, \varphi') \neq \perp \wedge \mu \sqsubseteq_R \mu' \quad \text{for some } \mu' \in Sat(\varphi') \end{aligned} \quad (\text{A.1})$$

We prove that for each $\varphi' \in C(S')$ there exists a corresponding $\varphi'' \in C(S'')$ such that $L'(s', a, \varphi') = L''(\nu(s'), a, \varphi'')$; and for each $\mu' \in Sat(\varphi')$ there is a corresponding $\mu'' \in Sat(\varphi'')$ such that for all $\mu \in Dist(S)$ if $\mu \sqsubseteq_R \mu'$, then $\mu \sqsubseteq_{R'} \mu''$.

As $s'' = \nu(s')$, then according to Definition 4.3.8 for each $\varphi' \in C(S')$, there exists a $\varphi'' \in C(S'')$, and vice versa, such that for all $t \in T$ and $v'' \in S''$:

$$\begin{aligned} L'(s', a, \varphi') = L''(\nu(s'), a, \varphi'') \quad \text{and,} \\ \exists \mu' \in Sat(\varphi') \text{ iff } \exists \mu'' \in Sat(\varphi'') : \mu'(t) = 0 \wedge \mu''(v'') = \mu'(v'') \end{aligned} \quad (\text{A.2})$$

Next we prove that every inconsistent state in T is not in a satisfaction relation R with any state in P , i.e., $T \cap Ran(R) = \emptyset$. By contraposition, let us suppose that there exists $t \in T \cap Ran(R)$. It means that there exists some $u \in S$ such that uRt , i.e., both u and t fulfill the requirements of Definition 4.3.4. But in this case t does not fulfill the requirements of Definition 4.3.7, i.e., t has at least one $p \in AP'$ such that $V'(t, p) \neq \perp$; and in case of every *must* a -transition $L'(t, a, \varphi')$ from t the satisfaction set $Sat(\varphi')$ is not empty. This means that $\nu(t) \neq \perp$ and $t \notin T$. Contradiction.

Moreover, as $S'' \subset S'$, therefore, $\nu(s') = s'$ and, hence, $R = R'$. This shows that $\mu \sqsubseteq_{R'} \mu''$. Hence, from Equations A.1 and A.2 we conclude that (1b) of Definition 4.3.4 also holds for relation R' .

Finally, R' is a satisfaction relation such that $s_0 R' s_0''$, thus $P \models \beta(M)$.

“ \Leftarrow ” Conversely, the steps of the proof go along the same lines as in the forward direction.

As the state space of M is finite, therefore, the fixpoint of β exists. Moreover, we have proved that β is implementation-conservative, thus the fixpoint of β verifies the same property. \square

Proof of Theorem 2

Theorem. For any ACPA N and abstraction function α :

$$N \preceq \alpha(N)$$

Proof. Let $N = (S, A, L, AP, V, s_0)$ be an ACPA and let $\alpha : S \rightarrow S'$ be an abstraction function such that $\alpha(N) = (S', A', L', AP', V', s'_0)$ is the induced ACPA. To prove that $N \preceq \alpha(N)$ we define a relation $R \subseteq S \times S'$ as:

$$R = \{(s, \alpha(s)) \mid s \in S\}$$

and show that R fulfills (1a), (1b), (2a) and (2b) of Definition 4.3.12.

1a. Assume R is a refinement relation, then for all $a \in A$ and sRs' , we have according to Definition 4.3.12:

$$\begin{aligned} \forall \varphi' \in C(S') : L'(s', a, \varphi') = \top \\ \implies \exists \varphi \in C(S) : L(s, a, \varphi) = \top \wedge \varphi \sqsubseteq_R \varphi' \quad \text{(A.3)} \end{aligned}$$

As sRs' and $L'(s', a, \varphi') = \top$, we have according to item (a) of Definition 4.3.10:

$$Sat(\varphi') = \alpha\left(\bigcap_{s \in \gamma(s'), \exists \varphi \in C(S) : L(s, a, \varphi) = \top} Sat(\varphi)\right)$$

As $s \in \gamma(s')$, therefore, there exists $\varphi \in C(S)$ such that $L(s, a, \varphi) = \top$. Now let $Sat(\varphi) = \left(\bigcap_{s \in \gamma(s'), \exists \varphi \in C(S) : L(s, a, \varphi) = \top} Sat(\varphi)\right)$ such that $Sat(\varphi') = \alpha(Sat(\varphi))$. Therefore, for each $\mu \in Sat(\varphi)$, there exists $\mu' \in Sat(\varphi')$ such that $\mu' = \alpha(\mu)$, hence, $\mu \sqsubseteq_R \mu'$. Using Definition 4.3.11, it is proved that Equation A.3 holds.

1b. Again assume R is a refinement relation, then for all $a \in A$ and sRs' , we have according to Definition 4.3.12:

$$\begin{aligned} \forall \varphi \in C(S) : L(s, a, \varphi) \neq \perp \\ \implies \exists \varphi' \in C(S') : L'(s', a, \varphi') \neq \perp \wedge \varphi \sqsubseteq_R \varphi' \quad \text{(A.4)} \end{aligned}$$

We have four possible cases:

1. $L(s, a, \varphi) = \top$ and $L'(s', a, \varphi') = \top$ satisfying item (a) of Definition 4.3.10.
2. $L(s, a, \varphi) = \top$ and $L'(s', a, \varphi') = ?$ satisfying item (b1) of Definition 4.3.10.

3. $L(s, a, \varphi) = ?$ and $L'(s', a, \varphi') = ?$ satisfying item (b2) of Definition 4.3.10.

4. $L(s, a, \varphi) = \top$ and $L'(s', a, \varphi') = ?$ satisfying item (b2) of Definition 4.3.10.

Case 4 can be proved in the same way as case 3. To prove case 3, we can write Equation A.4 as:

$$\begin{aligned} \forall \varphi \in C(S) : L(s, a, \varphi) = ? \\ \implies \exists \varphi' \in C(S') : L'(s', a, \varphi') = ? \wedge \varphi \sqsubseteq_R \varphi' \quad \text{(A.5)} \end{aligned}$$

As sRs' and $L'(s', a, \varphi') = ?$, we have according to item (b2) of Definition 4.3.10:

$$Sat(\varphi') = \alpha\left(\bigcup_{s \in \gamma(s'), \exists \varphi \in C(S) : L(s, a, \varphi) \neq \perp} Sat(\varphi)\right)$$

As $s \in \gamma(s')$, for all $\varphi \in C(S)$ having $L(s, a, \varphi) = ?$, it implies that $Sat(\varphi) \subseteq \bigcup_{s \in \gamma(s'), \exists \varphi \in C(S) : L(s, a, \varphi) \neq \perp} Sat(\varphi)$. Therefore, for each $\mu \in Sat(\varphi)$, there exists $\mu' \in Sat(\varphi')$ such that $\mu' = \alpha(\mu)$, hence, $\mu \sqsubseteq_R \mu'$. Using Definition 4.3.11, it is proved that Equation A.5 holds.

Now we prove case 1 and 2. In Equation A.4, every constraint function $\varphi \in C(S)$, having $L(s, a, \varphi) = \top$, is split into two constraint functions φ_1 and φ_2 as:

$$\begin{aligned} Sat(\varphi_1) &= \bigcap_{s \in \gamma(s'), \exists \varphi \in C(S) : L(s, a, \varphi) = \top} Sat(\varphi) \\ Sat(\varphi_2) &= Sat(\varphi) \setminus Sat(\varphi_1) \end{aligned}$$

and, as a result, Equation A.4 is also split into two equations, which represent case 1 and 2:

$$\begin{aligned} L(s, a, \varphi_1) = \top \\ \implies \exists \varphi' \in C(S') : L'(s', a, \varphi') = \top \wedge \varphi_1 \sqsubseteq_R \varphi' \quad \text{(A.6)} \end{aligned}$$

$$\begin{aligned} L(s, a, \varphi_2) = \top \\ \implies \exists \varphi' \in C(S') : L'(s', a, \varphi') = ? \wedge \varphi_2 \sqsubseteq_R \varphi' \quad \text{(A.7)} \end{aligned}$$

The proofs of Equations A.6 and A.7 go along the same lines as that of Equations A.3 and A.5 respectively. The proofs of items (2a) and (2b) of Definition 4.3.12 are trivial. \square

Proof of Theorem 3

Theorem. For ACPA N and N' :

$$N \preceq N' \implies \llbracket N \rrbracket \subseteq \llbracket N' \rrbracket$$

Proof. It follows trivially from Definitions 4.3.4 and 4.3.12. \square

Proof of Theorem 4

Theorem. Let M , N , O and P be ACPA, then:

$$(M \preceq N) \wedge (O \preceq P) \implies (M \wedge O) \preceq (N \wedge P)$$

Proof. Let $M = (S, A, L, AP, V, s_0)$, $N = (S', A', L', AP', V', s'_0)$, $O = (S'', A'', L'', AP'', V'', s''_0)$ and $P = (S''', A''', L''', AP''', V''', s'''_0)$ be ACPA such that $M \preceq N$ and $O \preceq P$. Without loss of generality we assume $A = A'$ and $A'' = A'''$. The conjunction of M and O is given as $M \wedge O = (S \times S'', A \cap A'', \tilde{L}, AP \cap AP'', \tilde{V}, (s_0, s''_0))$ and that of N and P as $N \wedge P = (S' \times S''', A \cap A'', \tilde{L}', AP' \cap AP''', \tilde{V}', (s'_0, s'''_0))$. Let $R \subseteq S \times S'$ and $R' \subseteq S'' \times S'''$ be refinement relations with $s_0 R s'_0$ and $s''_0 R' s'''_0$ respectively. We define a relation $\tilde{R} \subseteq (S \times S'') \times (S' \times S''')$ as:

$$\tilde{R} = \{((s, s''), (s', s''')) \mid s R s' \text{ and } s'' R' s'''\}$$

and show that it fulfills (1a), (1b), (2a) and (2b) of Definition 4.3.12. Let $\varphi \in C(S)$, $\varphi' \in C(S')$, $\varphi'' \in C(S'')$, $\varphi''' \in C(S''')$, $\tilde{\varphi} \in C(S \times S'')$ and $\tilde{\varphi}' \in C(S' \times S''')$.

We show the proof for item (1a). Items (1b), (2a) and (2b) of Definition 4.3.12 can be proved in the same way as (1a).

1a. Let $(s, s'') \tilde{R} (s', s''')$ such that $s R s'$ and $s'' R' s'''$. As $s R s'$, then for all $a \in A$ we have by Definition 4.3.12:

$$\forall \varphi' : L'(s', a, \varphi') = \top \implies \exists \varphi : L(s, a, \varphi) = \top \wedge \varphi \sqsubseteq_R \varphi'$$

(A.8)

Similarly, as $s'' R' s'''$, for all $a \in A''$ we have:

$$\forall \varphi''' : L'''(s''', a, \varphi''') = \top \implies \exists \varphi'' : L''(s'', a, \varphi'') = \top \wedge \varphi'' \sqsubseteq_{R'} \varphi'''$$

(A.9)

Now assume \tilde{R} is a refinement relation, then for all $a \in A \cap A''$ and $(s, s'') \tilde{R} (s', s''')$ we have:

$$\forall \tilde{\varphi}' : \tilde{L}'((s', s'''), a, \tilde{\varphi}') = \top \implies \exists \tilde{\varphi} : \tilde{L}((s, s''), a, \tilde{\varphi}) = \top \wedge \tilde{\varphi} \sqsubseteq_{\tilde{R}} \tilde{\varphi}'$$

As states (s', s''') and (s, s'') are results of conjunction operations, therefore, according to Definition 4.3.13:

$$\begin{aligned} \forall \varphi', \forall \varphi''' : L'(s', a, \varphi') \sqcup L'''(s''', a, \varphi''') = \top \\ \implies \exists \varphi, \exists \varphi'' : L(s, a, \varphi) \sqcup L''(s'', a, \varphi'') = \top \wedge \varphi \cdot \varphi'' \sqsubseteq_{\tilde{R}} \varphi' \cdot \varphi''' \end{aligned} \quad \text{(A.10)}$$

Along with Lemma 7, Equation A.10 follows directly from Equations A.8 and A.9. \square

Proof of Theorem 5

Theorem. *Refinement \preceq is a precongruence w.r.t. $\|\bar{A}$.*

Proof. We prove *reflexivity* and *transitivity* of \preceq relation as well as preservation in the context of $\|\bar{A}$.

Reflexivity: Reflexivity of \preceq follows trivially from Definition 4.3.12.

Transitivity: Let $M = (S, A, L, AP, V, s_0)$, $N = (S', A', L', AP', V', s'_0)$ and $P = (S'', A'', L'', AP'', V'', s''_0)$ be ACPA. To argue about refinement of states in different specifications we have to analyze their disjoint union. For simplicity in this proof, we refrain from explicitly doing so. However, we assume that the action sets of all three models are same i.e. $A = A' = A''$. Let $R \subseteq S \times S'$ and $R' \subseteq S' \times S''$ be refinement relations with $s_0 R s'_0$ and $s'_0 R' s''_0$ respectively. We define a relation $R'' \subseteq S \times S''$ as:

$$R'' = \{(s, s'') \mid \exists s' \in S' : (s, s') \in R \text{ and } (s', s'') \in R'\}$$

and show that it fulfills (1a), (1b), (2a) and (2b) of Definition 4.3.12. We show the proofs for items (1a) and (1b). Items (2a) and (2b) can be proved in the same way as (1a) and (1b).

As $(s, s') \in R$, by Definition 4.3.12 we have:

$$\begin{aligned} \forall a \in A, \forall \varphi' \in C(S') : L'(s', a, \varphi') = \top \\ \implies \exists \varphi \in C(S) : L(s, a, \varphi) = \top \wedge \varphi \sqsubseteq_R \varphi' \end{aligned} \quad \text{(A.11)}$$

$$\begin{aligned} \forall a \in A, \forall \varphi \in C(S) : L(s, a, \varphi) \neq \perp \\ \implies \exists \varphi' \in C(S') : L'(s', a, \varphi') \neq \perp \wedge \varphi \sqsubseteq_R \varphi' \end{aligned} \quad \text{(A.12)}$$

and similarly, as $(s', s'') \in R'$, by Definition 4.3.12 we have:

$$\begin{aligned} \forall a \in A, \forall \varphi'' \in C(S'') : L''(s'', a, \varphi'') = \top \\ \implies \exists \varphi' \in C(S') : L'(s', a, \varphi') = \top \wedge \varphi' \sqsubseteq_{R'} \varphi'' \end{aligned} \quad \text{(A.13)}$$

$$\begin{aligned} \forall a \in A, \forall \varphi' \in C(S') : L'(s', a, \varphi') \neq \perp \\ \implies \exists \varphi'' \in C(S'') : L''(s'', a, \varphi'') \neq \perp \wedge \varphi' \sqsubseteq_{R'} \varphi'' \end{aligned} \quad \text{(A.14)}$$

It follows from Equations A.11 and A.13 that:

$$\begin{aligned} \forall a \in A, \forall \varphi'' \in C(S'') : L''(s'', a, \varphi'') = \top \\ \implies \exists \varphi' \in \text{Dist}(S'), \exists \varphi \in \text{Dist}(S) : L(s, a, \varphi) = \top \wedge \varphi \sqsubseteq_R \varphi' \wedge \varphi' \sqsubseteq_{R'} \varphi'' \end{aligned}$$

and using Lemma 6,

$$\begin{aligned} \forall a \in A, \forall \varphi'' \in C(S'') : L''(s'', a, \varphi'') = \top \\ \implies \exists \varphi \in C(S) : L(s, a, \varphi) = \top \wedge \varphi \sqsubseteq_{R''} \varphi'' \end{aligned}$$

Similarly, it follows from A.12 and A.14 that:

$$\begin{aligned} \forall a \in A, \forall \varphi \in C(S) : L(s, a, \varphi) \neq \perp \\ \implies \exists \varphi' \in C(S'), \exists \varphi'' \in C(S'') : L''(s'', a, \varphi'') \neq \perp \\ \wedge \varphi \sqsubseteq_R \varphi' \wedge \varphi' \sqsubseteq_{R'} \varphi'' \end{aligned}$$

and using Lemma 6,

$$\begin{aligned} \forall a \in A, \forall \varphi \in C(S) : L(s, a, \varphi) \neq \perp \\ \implies \exists \varphi'' \in C(S'') : L''(s'', a, \varphi'') \neq \perp \\ \wedge \varphi \sqsubseteq_{R''} \varphi'' \end{aligned}$$

Now we prove that the parallel composition of two ACPA preserves refinement relations.

Let $M = (S, A, L, AP, V, s_0)$, $N = (S', A', L', AP', V', s'_0)$,

$O = (S'', A'', L'', AP'', V'', s_0'')$ and $P = (S''', A''', L''', AP''', V''', s_0''')$ be ACPA such that $M \preceq N$ and $O \preceq P$. Without loss of generality we assume $A = A'$ and $A'' = A'''$. Let $\bar{A} \subseteq A \cap A''$ be the set of synchronization actions. The parallel composition of M and O is given as $M \parallel_{\bar{A}} O = (S \times S'', A \cup A'', \tilde{L}, AP \cup AP'', \tilde{V}, (s_0, s_0''))$ and the that of N and P as $N \parallel_{\bar{A}} P = (S' \times S''', A \cup A''', \tilde{L}', AP' \cup AP''', \tilde{V}', (s_0', s_0'''))$. We prove that:

$$M \preceq N \wedge O \preceq P \implies M \parallel_{\bar{A}} O \preceq N \parallel_{\bar{A}} P$$

Let $R \subseteq S \times S'$ and $R' \subseteq S'' \times S'''$ be refinement relations with $s_0 R s_0'$ and $s_0'' R' s_0'''$ respectively. We define a relation $\tilde{R} \subseteq (S \times S'') \times (S' \times S''')$ as:

$$\tilde{R} = \{((s, s''), (s', s''')) \mid s R s' \text{ and } s'' R' s'''\}$$

and show that it fulfills (1a), (1b), (2a) and (2b) of Definition 4.3.12. Let $\varphi \in C(S)$, $\varphi' \in C(S')$, $\varphi'' \in C(S'')$, $\varphi''' \in C(S''')$, $\tilde{\varphi} \in C(S \times S'')$ and $\tilde{\varphi}' \in C(S' \times S''')$.

We show the proof for item (1a). Items (1b), (2a) and (2b) of Definition 4.3.12 can be shown in a similar fashion as item (1a).

1a. First we discuss *synchronous* case:

For $a \in \bar{A}$, let $(s, s'') \tilde{R} (s', s''')$ such that $s R s'$ and $s'' R' s'''$. As $s R s'$, then by Definition 4.3.12 we have:

$$\forall \varphi' : L'(s', a, \varphi') = \top \implies \exists \varphi : L(s, a, \varphi) = \top \wedge \varphi \sqsubseteq_R \varphi'$$

(A.15)

Similarly, as $s'' R' s'''$, we have:

$$\forall \varphi''' : L'''(s''', a, \varphi''') = \top \implies \exists \varphi'' : L''(s'', a, \varphi'') = \top \wedge \varphi'' \sqsubseteq_{R'} \varphi'''$$

(A.16)

Now assume \tilde{R} is a refinement relation, then $(s, s'')\tilde{R}(s', s''')$ can be written as:

$$\forall \tilde{\varphi}' : \tilde{L}'((s', s'''), a, \tilde{\varphi}') = \top \implies \exists \tilde{\varphi} : \tilde{L}((s, s''), a, \tilde{\varphi}) = \top \wedge \tilde{\varphi} \sqsubseteq_{\tilde{R}} \tilde{\varphi}'$$

As a is a synchronization action, it follows according to Definition 4.3.14:

$$\begin{aligned} \forall \varphi', \forall \varphi''' : L'(s', a, \varphi') \sqcap L'''(s''', a, \varphi''') = \top \\ \implies \exists \varphi, \exists \varphi'' : L(s, a, \varphi) \sqcap L''(s'', a, \varphi'') = \top \wedge \varphi \cdot \varphi'' \sqsubseteq_{\tilde{R}} \varphi' \cdot \varphi''' \\ \iff \\ \forall \varphi', \forall \varphi''' : L'(s', a, \varphi') \wedge L'''(s''', a, \varphi''') = \top \\ \implies \exists \varphi, \exists \varphi'' : L(s, a, \varphi) = \top \wedge L''(s'', a, \varphi'') = \top \\ \wedge \varphi \cdot \varphi'' \sqsubseteq_{\tilde{R}} \varphi' \cdot \varphi''' \end{aligned}$$

(A.17)

Along with Lemma 7, Equation A.17 follows directly from Equations A.15 and A.16.

Next we prove that the refinement relation is also preserved in the *asynchronous* case. Without loss of generality we assume $a \in A \setminus \bar{A}$.

For $a \in A \setminus \bar{A}$, assume \tilde{R} is a refinement relation, then for all $(s, s'')\tilde{R}(s', s''')$, we have by Definition 4.3.12:

$$\forall \tilde{\varphi}' : \tilde{L}'((s', s'''), a, \tilde{\varphi}') = \top \implies \exists \tilde{\varphi} : \tilde{L}((s, s''), a, \tilde{\varphi}) = \top \wedge \tilde{\varphi} \sqsubseteq_{\tilde{R}} \tilde{\varphi}'$$

As $a \in A \setminus \bar{A}$, then by Definition 4.3.14, we can write:

$$\forall \varphi' : L'(s', a, \varphi') = \top \implies \exists \varphi : L(s, a, \varphi) = \top \wedge \varphi \cdot I(s'', \cdot) \sqsubseteq_{\tilde{R}} \varphi' \cdot I(s''', \cdot)$$

(A.18)

Equation A.18 follows directly from Equation A.15 along with Lemma 8. Similarly, (2a) and (2b) of Definition 4.3.12 can be proved.

□

Proof of Theorem 6

Theorem. *Let M and N be ACPA, \bar{A} a synchronization set, and α_1 , α_2 and α_3 be abstraction functions such that $\alpha_3 = \alpha_1 \times \alpha_2$, then:*

$$\alpha_1(M) \parallel_{\bar{A}} \alpha_2(N) = \alpha_3(M \parallel_{\bar{A}} N) \quad \text{up to isomorphism}$$

Proof. Let $M = (S, A, L, AP, V, s_0)$ and $N = (S'', A'', L'', AP'', V'', s''_0)$ be ACPA and $\bar{A} \subseteq A \cap A''$ a set of synchronization actions such that the parallel composition of M and N is given as $M \parallel_{\bar{A}} N = (S \times S'', A \cup A'', \tilde{L}, AP \cup AP'', \tilde{V}, (s_0, s''_0))$. Let $\alpha_1 : S \rightarrow S'$ and $\alpha_2 : S'' \rightarrow S'''$ be the abstraction functions such that the abstraction function α_3 is given as: $\alpha_3 : S \times S'' \rightarrow \alpha_1(S) \times \alpha_2(S'')$. Let $\alpha_1(M) = (S', A, L', AP', V', s'_0)$, $\alpha_2(N) = (S''', A'', L''', AP'', V''', s'''_0)$ and $\alpha_3(M \parallel_{\bar{A}} N) = (S' \times S''', A \cup A'', \tilde{L}', AP \cup AP'', \tilde{V}', (s'_0, s'''_0))$ be the induced ACPA. Let $\varphi \in C(S)$, $\varphi' \in C(S')$, $\varphi'' \in C(S'')$, $\varphi''' \in C(S''')$, $\tilde{\varphi} \in C(S \times S'')$, $\tilde{\varphi}' \in C(S' \times S''')$, $s \in S$, $s' \in S'$, $s'' \in S''$, $s''' \in S'''$, $(s, s'') \in S \times S''$ and $(s', s''') \in S' \times S'''$. Let $\gamma_1 = \alpha_1^{-1}$, $\gamma_2 = \alpha_2^{-1}$ and $\gamma_3 = \alpha_3^{-1}$.

“ \implies ” First we prove the *synchronous* case. Items (a), (b2) and (c) of Definition 4.3.10 can be proved in the same way as item (b1). However, in the proof of item (b1) we also need to refer to item (a) of Definition 4.3.10. Therefore, we mention item (a) for both $\alpha_1(M)$ and $\alpha_2(N)$, and we use it later in the proof.

(b1). In $\alpha_1(M)$, for all $a \in \bar{A}$, $\varphi' \in C(S')$ and $s' \in S'$, we have according to Definition 4.3.10:

$$\begin{aligned} \forall s \in \gamma_1(s') : \exists \varphi : L(s, a, \varphi) = \top \wedge \\ \text{Sat}(\varphi') = \alpha_1 \left(\bigcap_{s \in \gamma_1(s'), \exists \varphi : L(s, a, \varphi) = \top} \text{Sat}(\varphi) \right) \\ \implies L'(s', a, \varphi') = \top \end{aligned} \tag{A.19}$$

$$\begin{aligned} \forall s \in \gamma_1(s') : \exists \varphi : L(s, a, \varphi) = \top \wedge \\ \text{Sat}(\varphi') = \alpha_1 \left(\bigcup_{s \in \gamma_1(s'), \exists \varphi : L(s, a, \varphi) = \top} \text{Sat}(\varphi) \right) \setminus \\ \alpha_1 \left(\bigcap_{s \in \gamma_1(s'), \exists \varphi : L(s, a, \varphi) = \top} \text{Sat}(\varphi) \right) \\ \implies L'(s', a, \varphi') = ? \end{aligned} \tag{A.20}$$

As a is a synchronization action, for all $\varphi' \in C(S')$ and $\varphi''' \in C(S''')$, it follows according to Definition 4.3.14:

$$\begin{aligned}
 & \forall s \in \gamma_1(s'), \forall s'' \in \gamma_2(s''') : \exists \varphi, \exists \varphi'' : L(s, a, \varphi) \sqcap L''(s'', a, \varphi'') = \top \wedge \\
 & \text{Sat}(\varphi' \cdot \varphi''') = \text{Sat}(\varphi') \cdot \text{Sat}(\varphi''') = \\
 & \alpha_1 \times \alpha_2 \left(\bigcup_{s \in \gamma_1(s'), s'' \in \gamma_2(s'''), \exists \varphi, \exists \varphi'' : L(s, a, \varphi) \sqcap L''(s'', a, \varphi'') = \top} \text{Sat}(\varphi \cdot \varphi'') \right) \\
 & \setminus \alpha_1 \times \alpha_2 \left(\bigcap_{s \in \gamma_1(s'), s'' \in \gamma_2(s'''), \exists \varphi, \exists \varphi'' : L(s, a, \varphi) \sqcap L''(s'', a, \varphi'') = \top} \text{Sat}(\varphi \cdot \varphi'') \right) \\
 & \implies L'(s', a, \varphi') \sqcap L'''(s''', a, \varphi''') = ? \\
 & \iff
 \end{aligned}$$

$$\begin{aligned}
 & \forall s \in \gamma_1(s'), \forall s'' \in \gamma_2(s''') : \exists \varphi, \exists \varphi'' : L(s, a, \varphi) = L''(s'', a, \varphi'') = \top \wedge \\
 & \text{Sat}(\varphi' \cdot \varphi''') = \text{Sat}(\varphi') \cdot \text{Sat}(\varphi''') = \\
 & \alpha_1 \left(\bigcup_{s \in \gamma_1(s'), \exists \varphi : L(s, a, \varphi) = \top} \text{Sat}(\varphi) \right) \cdot \alpha_2 \left(\bigcup_{s'' \in \gamma_2(s'''), \exists \varphi'' : L''(s'', a, \varphi'') = \top} \text{Sat}(\varphi'') \right) \\
 & \setminus \alpha_1 \left(\bigcap_{s \in \gamma_1(s'), \exists \varphi : L(s, a, \varphi) = \top} \text{Sat}(\varphi) \right) \cdot \alpha_2 \left(\bigcap_{s'' \in \gamma_2(s'''), \exists \varphi'' : L''(s'', a, \varphi'') = \top} \text{Sat}(\varphi'') \right) \\
 & \implies L'(s', a, \varphi') \sqcap L'''(s''', a, \varphi''') = ?
 \end{aligned}$$

(A.23)

Note that in Equation A.23 $L'(s', a, \varphi') \sqcap L'''(s''', a, \varphi''') = ?$. This is because either $L'(s', a, \varphi') = ?$ or $L'''(s''', a, \varphi''') = ?$. If both $L'(s', a, \varphi') = \top$ and $L'''(s''', a, \varphi''') = \top$, then it refers to part (a) of Definition 4.3.10. Let

$$\begin{aligned}
 P &= \bigcup_{s \in \gamma_1(s'), \exists \varphi : L(s, a, \varphi) = \top} \text{Sat}(\varphi), \\
 Q &= \bigcup_{s'' \in \gamma_2(s'''), \exists \varphi'' : L''(s'', a, \varphi'') = \top} \text{Sat}(\varphi''), \\
 X &= \bigcap_{s \in \gamma_1(s'), \exists \varphi : L(s, a, \varphi) = \top} \text{Sat}(\varphi), \text{ and} \\
 Y &= \bigcap_{s'' \in \gamma_2(s'''), \exists \varphi'' : L''(s'', a, \varphi'') = \top} \text{Sat}(\varphi'')
 \end{aligned}$$

Equations A.19 and A.20 then simplify as:

$$\begin{aligned}
 \forall s \in \gamma_1(s') : \exists \varphi : L(s, a, \varphi) = \top \wedge \text{Sat}(\varphi') &= \alpha_1(X) \\
 \implies L'(s', a, \varphi') &= \top
 \end{aligned}$$

(A.24)

$$\begin{aligned} \forall s \in \gamma_1(s') : \exists \varphi : L(s, a, \varphi) = \top \wedge \text{Sat}(\varphi') = \alpha_1(P) \setminus \alpha_1(X) \\ \implies L'(s', a, \varphi') = ? \end{aligned}$$

(A.25)

Similarly, Equations A.21 and A.22 simplify as:

$$\begin{aligned} \forall s'' \in \gamma_2(s''') : \exists \varphi'' : L''(s'', a, \varphi'') = \top \wedge \text{Sat}(\varphi''') = \alpha_2(Y) \\ \implies L'''(s''', a, \varphi''') = \top \end{aligned}$$

(A.26)

$$\begin{aligned} \forall s'' \in \gamma_2(s''') : \exists \varphi'' : L''(s'', a, \varphi'') = \top \wedge \text{Sat}(\varphi''') = \alpha_2(Q) \setminus \alpha_2(Y) \\ \implies L'''(s''', a, \varphi''') = ? \end{aligned}$$

(A.27)

Moreover, Equation A.23 simplifies as:

$$\begin{aligned} \forall s \in \gamma_1(s'), \forall s'' \in \gamma_2(s''') : \exists \varphi, \exists \varphi'' : L(s, a, \varphi) = L''(s'', a, \varphi'') = \top \wedge \\ \text{Sat}(\varphi' \cdot \varphi''') = \text{Sat}(\varphi') \cdot \text{Sat}(\varphi''') = \alpha_1(P) \cdot \alpha_2(Q) \setminus \alpha_1(X) \cdot \alpha_2(Y) \\ \implies L'(s', a, \varphi') \sqcap L'''(s''', a, \varphi''') = ? \end{aligned}$$

(A.28)

There are three possible cases: (1) $X = Y = \emptyset$, (2) $X = \emptyset$ or $Y = \emptyset$, and (3) $X \neq \emptyset$ and $Y \neq \emptyset$. We discuss each case separately.

(1). As $X = Y = \emptyset$, Equation A.28 then simplifies as:

$$\begin{aligned} \forall s \in \gamma_1(s'), \forall s'' \in \gamma_2(s''') : \exists \varphi, \exists \varphi'' : L(s, a, \varphi) = L''(s'', a, \varphi'') = \top \wedge \\ \text{Sat}(\varphi' \cdot \varphi''') = \text{Sat}(\varphi') \cdot \text{Sat}(\varphi''') = \alpha_1(P) \cdot \alpha_2(Q) \\ \implies L'(s', a, \varphi') \sqcap L'''(s''', a, \varphi''') = ? \end{aligned}$$

It then follows directly from Equations A.25 and A.27.

(2) Let $Y = \emptyset$, then $\alpha_1(P) \cdot \alpha_2(Q) \setminus \alpha_1(X) \cdot \alpha_2(Y)$ in Equation A.28 is reduced as:

$$\begin{aligned} \alpha_1(P) \cdot \alpha_2(Q) \setminus \alpha_1(X) \cdot \alpha_2(Y) &= \alpha_1(P) \cdot \alpha_2(Q) \\ &= (\alpha_1(P) \setminus \alpha_1(X) \cup \alpha_1(X)) \cdot \alpha_2(Q) \\ &= (\alpha_1(P) \setminus \alpha_1(X)) \cdot \alpha_2(Q) \cup \alpha_1(X) \cdot \alpha_2(Q) \end{aligned}$$

Equation A.28, then simplifies as:

$$\begin{aligned}
 & \forall s \in \gamma_1(s'), \forall s'' \in \gamma_2(s''') : \exists \varphi, \exists \varphi'' : L(s, a, \varphi) = L''(s'', a, \varphi'') = \top \wedge \\
 & Sat(\varphi' \cdot \varphi''') = Sat(\varphi') \cdot Sat(\varphi''') = (\alpha_1(P) \setminus \alpha_1(X)) \cdot \alpha_2(Q) \cup \alpha_1(X) \cdot \alpha_2(Q) \\
 & \implies L'(s', a, \varphi') \sqcap L'''(s''', a, \varphi''') = ?
 \end{aligned}$$

It then follows directly from Equations A.24, A.25 and A.27.

(3). As both X and Y are not empty, then $\alpha_1(P) \cdot \alpha_2(Q) \setminus \alpha_1(X) \cdot \alpha_2(Y)$ in Equation A.28 is reduced as:

$$\begin{aligned}
 & (\alpha_1(P) \cdot \alpha_2(Q)) \setminus (\alpha_1(X) \cdot \alpha_2(Y)) \\
 & = ((\alpha_1(P) \setminus \alpha_1(X)) \cup \alpha_1(X)) \cdot (\alpha_2(Q) \setminus \alpha_2(Y) \cup \alpha_2(Y)) \setminus (\alpha_1(X) \cdot \alpha_2(Y)) \\
 & = ((\alpha_1(P) \setminus \alpha_1(X)) \cdot (\alpha_2(Q) \setminus \alpha_2(Y)) \cup (\alpha_1(P) \setminus \alpha_1(X)) \cdot \alpha_2(Y) \cup \\
 & \quad \alpha_1(X) \cdot (\alpha_2(Q) \setminus \alpha_2(Y)) \cup \alpha_1(X) \cdot \alpha_2(Y)) \setminus (\alpha_1(X) \cdot \alpha_2(Y)) \\
 & = (\alpha_1(P) \setminus \alpha_1(X)) \cdot (\alpha_2(Q) \setminus \alpha_2(Y)) \cup (\alpha_1(P) \setminus \alpha_1(X)) \cdot \alpha_2(Y) \cup \alpha_1(X) \cdot (\alpha_2(Q) \setminus \alpha_2(Y))
 \end{aligned}$$

Equation A.28 then simplifies as:

$$\begin{aligned}
 & \forall s \in \gamma_1(s'), \forall s'' \in \gamma_2(s''') : \exists \varphi, \exists \varphi'' : L(s, a, \varphi) = L''(s'', a, \varphi'') = \top \wedge \\
 & Sat(\varphi' \cdot \varphi''') = Sat(\varphi') \cdot Sat(\varphi''') = \\
 & (\alpha_1(P) \setminus \alpha_1(X)) \cdot (\alpha_2(Q) \setminus \alpha_2(Y)) \cup (\alpha_1(P) \setminus \alpha_1(X)) \cdot \alpha_2(Y) \cup \alpha_1(X) \cdot (\alpha_2(Q) \setminus \alpha_2(Y)) \\
 & \implies L'(s', a, \varphi') \sqcap L'''(s''', a, \varphi''') = ?
 \end{aligned}$$

It then follows directly from Equations A.24, A.25, A.26 and A.27.

Next we prove that this implication also holds for *asynchronous* case. Without loss of generality, we assume $a \in A \setminus \bar{A}$. Items (a), (b2) and (c) of Definition 4.3.10 can be shown in a similar fashion as item (b1).

(b1). In the induced ACPA $\alpha_3(M \parallel_{\bar{A}} N)$, for all $a \in A \setminus \bar{A}$, $\tilde{\varphi}' \in C(S' \times S''')$ and $(s', s''') \in S' \times S'''$, we have according to Definition 4.3.10:

$$\begin{aligned}
 & \forall (s, s'') \in \gamma_3((s', s''')) : \exists \tilde{\varphi} : \tilde{L}((s, s''), a, \tilde{\varphi}) = \top \wedge \\
 & Sat(\tilde{\varphi}') = \alpha_3 \left(\bigcup_{(s, s'') \in \gamma_3((s', s''')), \exists \tilde{\varphi} : \tilde{L}((s, s''), a, \tilde{\varphi}) = \top} Sat(\tilde{\varphi}) \right) \setminus \\
 & \quad \alpha_3 \left(\bigcap_{(s, s'') \in \gamma_3((s', s''')), \exists \tilde{\varphi} : \tilde{L}((s, s''), a, \tilde{\varphi}) = \top} Sat(\tilde{\varphi}) \right) \\
 & \implies \tilde{L}'((s', s'''), a, \tilde{\varphi}') = ?
 \end{aligned}$$

As $a \in A \setminus \bar{A}$, then for all $\varphi' \in C(S')$ it follows according to Definition 4.3.14:

$$\begin{aligned} & \forall s \in \gamma_1(s') : \exists \varphi : L(s, a, \varphi) = \top \wedge \text{Sat}(\varphi') \cdot I(s''', \cdot) = \\ & \alpha_1 \times \alpha_2 \left(\bigcup_{s \in \gamma_1(s'), s'' \in \gamma_2(s'''), \exists \varphi : L(s, a, \varphi) = \top} \text{Sat}(\varphi) \cdot I(s'', \cdot) \right) \\ & \setminus \alpha_1 \times \alpha_2 \left(\bigcap_{s \in \gamma_1(s'), s'' \in \gamma_2(s'''), \exists \varphi : L(s, a, \varphi) = \top} \text{Sat}(\varphi) \cdot I(s'', \cdot) \right) \\ & \implies L'(s', a, \varphi') = ? \end{aligned}$$

\iff

$$\begin{aligned} & \forall s \in \gamma_1(s') : \exists \varphi : L(s, a, \varphi) = \top \wedge \text{Sat}(\varphi') \cdot I(s''', \cdot) = \\ & \alpha_1 \left(\bigcup_{s \in \gamma_1(s'), \exists \varphi : L(s, a, \varphi) = \top} \text{Sat}(\varphi) \right) \cdot \alpha_2 \left(\bigcup_{s'' \in \gamma_2(s''')} I(s'', \cdot) \right) \\ & \setminus \alpha_1 \left(\bigcap_{s \in \gamma_1(s'), \exists \varphi : L(s, a, \varphi) = \top} \text{Sat}(\varphi) \right) \cdot \alpha_2 \left(\bigcap_{s'' \in \gamma_2(s''')} I(s'', \cdot) \right) \\ & \implies L'(s', a, \varphi') = ? \end{aligned}$$

\iff

$$\begin{aligned} & \forall s \in \gamma_1(s') : \exists \varphi : L(s, a, \varphi) = \top \wedge \text{Sat}(\varphi') \cdot I(s''', \cdot) = \\ & \alpha_1 \left(\bigcup_{s \in \gamma_1(s'), \exists \varphi : L(s, a, \varphi) = \top} \text{Sat}(\varphi) \right) \cdot \alpha_2 \left(\{ I(s'', \cdot) \mid s'' \in \gamma_2(s''') \} \right) \\ & \setminus \alpha_1 \left(\bigcap_{s \in \gamma_1(s'), \exists \varphi : L(s, a, \varphi) = \top} \text{Sat}(\varphi) \right) \cdot \alpha_2 \left(\{ I(s'', \cdot) \mid s'' \in \gamma_2(s''') \} \right) \\ & \implies L'(s', a, \varphi') = ? \end{aligned}$$

\iff

$$\begin{aligned} & \forall s \in \gamma_1(s') : \exists \varphi : L(s, a, \varphi) = \top \wedge \text{Sat}(\varphi') \cdot I(s''', \cdot) = \\ & \alpha_1 \left(\bigcup_{s \in \gamma_1(s'), \exists \varphi : L(s, a, \varphi) = \top} \text{Sat}(\varphi) \right) \cdot I(s''', \cdot) \\ & \setminus \alpha_1 \left(\bigcap_{s \in \gamma_1(s'), \exists \varphi : L(s, a, \varphi) = \top} \text{Sat}(\varphi) \right) \cdot I(s''', \cdot) \\ & \implies L'(s', a, \varphi') = ? \end{aligned}$$

(A.29)

Equation A.29 follows directly from Equation A.20.

“ \Leftarrow ” Conversely, the steps of the proof go along the same lines as in the forward direction.

□

Bibliography

- [CDL⁺10] Benoit Caillaud, Benoit Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. Compositional design methodology with constraint Markov chains. 2010.
- [Fel68] William Feller. *An Introduction to Probability Theory and its Applications, Vol. I*. John Wiley & Sons, Inc., 1968.
- [Her02] Holger Hermanns. *Interactive Markov Chains and the Quest for Quantified Quality*, volume 2428 of *LNCS*. Springer, Berlin, 2002.
- [HJS01] Michael Huth, Radha Jagadeesan, and David Schmidt. Modal transition systems: A foundation for three-valued program analysis. *LNCS*, 2028:155–169, 2001.
- [JL91] Bengt Jonsson and Kim G. Larsen. Specification and refinement of probabilistic processes. In *Logic in Computer Science*, pages 266–277. IEEE Press, 1991.
- [KKN09] Joost-Pieter Katoen, Daniel Klink, and Martin R. Neuhäuser. Compositional abstraction for stochastic systems. In *International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS)*, volume 5813 of *LNCS*, pages 195–211. Springer, 2009.
- [LT88] Kim G. Larsen and Bent Thomsen. A modal process logic. In *IEEE Symposium on Logic in Computer Science (LICS)*, pages 203–210. IEEE Computer Society Press, 1988.
- [Seg95] Roberto Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995.