

Logic and Model Checking for Hidden Markov Models

Diplomarbeit

Lijun Zhang

Advisor: Prof. Dr.-Ing. Holger Hermanns and Dr. David N. Jansen

Dependable Systems and Software Group
Prof. Dr.-Ing. Holger Hermanns
Stuhlsatzenhausweg 85,
66123 Saarbrücken,
Germany

March 31, 2004

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen verwendet habe.

Saarbrücken, den 31. März 2004

Abstract

The branching-time temporal logic PCTL* has been introduced to specify quantitative properties over probability systems, such as discrete-time Markov chains. Until now, however, no logics have been defined to specify properties over hidden Markov models (HMMs). In HMM the states are hidden, and the hidden processes produce a sequence of observations. In this paper we extend the logic PCTL* to POCTL*. With our logic one can state properties such as “there is at least a 90 percent probability that the model produces a sequence of observations” over HMMs. Subsequently, we give algorithms for checking whether a state in a given hidden Markov model satisfies a formula in the logic POCTL*.

Acknowledgement

First of all, I want to thank Holger Hermanns, my supervisor. He nurtured my interest in Logic and Model Checking and made it possible that I could do some research in this field. He always pointed out to me the bright way when I staggered in the blind alley. His humor shortened the emotional distance between us and also eased the serious scientific atmosphere.

Next, I would like to express my gratitude to David N. Jansen. He was always there to offer me help. He was so amiable that I never had the feeling of being restricted to speak out my problems. Every time I was impressed upon seeing the returned paper of different phases from him, which should be read and checked with great patience and earnestness. Thank you.

Then, the thanks would go to my best friend here in Germany—Björn Wachter. We got to know each other from the first semester in Saarbrücken. We had taken a lot of courses together and became good partners in studying. He helped me unselfishly on many significant occasions.

The love and care from my parents are there without a stop from the very beginning, and cannot be simply dedicated to thanks. They bestowed me the purest human virtue and instructed me to face and conquer difficulties in life.

Finally, I want to give thanks to my fiancée who is always beside me, taking care of me and cheering me up.

Contents

1	Introduction	9
1.1	Overview	9
1.2	Contributions of This Thesis	10
1.3	Related Work	10
2	Preliminaries	13
2.1	Some Definitions	13
2.2	The Logic PCTL* and Its Sublogics	14
2.3	Constructing Büchi Automata from LTL Formulas	17
2.3.1	Normal Form LTL Formulas	18
2.3.2	Creating Graphs	18
2.3.3	The Generalized Büchi Automaton	20
2.3.4	The Büchi Automaton	21
2.4	Model Checking for PCTL* and Its Sublogics	22
3	Hidden Markov Models	27
3.1	Labeled Discrete-Time HMMs	27
3.2	Belief State	28
3.3	Induced DTMC	32
3.4	Paths in HMM and Probability Spaces over Paths	33
4	The Logic POCTL*	37
4.1	Syntax of POCTL*	37
4.2	Semantics of POCTL*	38
4.3	The Sublogics	39
4.4	Specifying Properties in POCTL*	41
5	Model Checking	43
5.1	POCTL* Formulas	43
5.2	POCTL Formulas	45
5.3	Constructing Büchi Automata from OLTL Formulas	45
5.3.1	Normal Form OLTL Formulas	46
5.3.2	Creating Graphs	46
5.3.3	The Generalized Büchi Automaton	48
5.3.4	The Bounded Until Formula	49
5.4	QOS Formulas	49
5.5	Improving the Efficiency	53
5.6	Complexity of the Model Checking Algorithms	55

6 Conclusion and Future Work	57
6.1 Conclusion	57
6.2 Future Work	57
A Proof of Theorem 4.2.2	59
B Proof of Theorem 5.3.2	61
C Counterexample	67

Chapter 1

Introduction

1.1 Overview

Hidden Markov models (HMMs) [33] were developed in the late 1960's and had been proven to be very important for many applications, especially speech recognition [25], character recognition [39, 36] and biological sequence analysis [11].

An HMM is a doubly embedded stochastic process with an underlying stochastic process (DTMC) that is *hidden*, but can only be observed through another set of stochastic processes that produce a sequence of observations. Given the sequence of observations, we do not exactly know the current state, but we do know the probability distribution over the set of states. Rabiner [33] considered three basic *objective functions* of interest for a given HMM \mathcal{H} :

1. Given the observation sequence $O = (o_0, o_1, \dots, o_n)$, how do we efficiently compute $P(O|\mathcal{H})$, the probability of the observation sequence?
2. Given the observation sequence $O = (o_0, o_1, \dots, o_n)$, how do we choose a corresponding state sequence (s_0, s_1, \dots, s_n) which is optimal in some meaningful sense (i.e., best “explains” the observations)?
3. How do we adjust the model parameters to maximize $P(O|\mathcal{H})$?

For an HMM, one wants to specify properties over the underlying process. In addition, one is also interested in reasoning about properties over the other set of stochastic processes which produce the observations. In this paper, we introduce a logic called POCTL* which allows us to specify properties over HMMs, e.g., properties related to the above problems (see Section 4.4). As an example we consider the property:

There is at least a 90 percent probability that the model produces a sequence of observations $O = (o_0, o_1, \dots, o_n)$.

This property can be expressed in POCTL* by $\mathcal{P}_{\geq 0.9}(\mathbf{X}_{o_0} \mathbf{X}_{o_1} \dots \mathbf{X}_{o_n} tt)$. As indicated by Rabiner [33], this probability can be viewed as the score which specifies how well a given model matches the observations. In *Speech Recognition* [25], we want to find out the most likely sentence (with the highest score) given a language and some acoustic input (observations). Assuming that we know that the HMM for the word “Need” produces the acoustic observations with probability at least 0.9, then we can almost conclude that this acoustic input represents the word “Need”.

On the one hand, POCTL* is basically an extension of PCTL* where the next operator is equipped with an observation constraint. On the other hand, POCTL* can be also considered as a variant of

the temporal logic ACTL*, presented by De Nicola *et al.* [30, 24], in which the usual next operator is extended to constrain the action label of the transition.

The PCTL* model checking [23, 2, 1, 20] problem can be reduced to the QLS (quantitative LTL specification) model checking problem. For QLS model checking, one constructs first a Büchi automaton for an LTL formula using well-known methods [41, 38, 19], and then builds the product of the system and the constructed Büchi automaton. Finally, the QLS model checking problem can be reduced to a reach probability analysis in the product system.

Following the same line, we shall present the POCTL* model checking algorithm as follows. First, it will be reduced to the QOS (quantitative OLTL specification) model checking problem. The latter can be further reduced to a reach probability analysis in the product automaton. To that end, we construct a Büchi automaton for a given OLTL formula. This construction is an adaption of the one presented by Gerth *et al.* [19].

1.2 Contributions of This Thesis

Summarizing, the main contributions of this thesis are:

- Define probability spaces for a given HMM.
- Introduce a branching-time temporal logic POCTL* which allows us to specify state-based, path-based and belief state-based properties over HMMs.
- Construct a Büchi automaton for an OLTL formula ϕ such that the Büchi automaton accepts exactly the infinite words satisfying ϕ .
- Present the model checking algorithms for POCTL* over HMMs.

1.3 Related Work

The basic model in Figure 1.1 is a Markov Model (MM) [22], some people also call it a Markov Process (MP). The time range in MMs can be either discrete or continuous, and this distinction provides us discrete-time Markov Chains (DTMCs) [22, 8, 2] and continuous-time Markov Chains (CTMCs) [7] respectively. For simplicity, we assume that time ranges over the set of natural numbers \mathbb{N} in DTMCs, and time ranges over the subset of nonnegative real numbers \mathbb{R}_+ in CTMCs.

The models in Figure 1.1 can be divided into discrete-time parts and continuous-time parts. The arrows (both solid and dashed) between the models M and N mean that M can be considered as a special case of N , both in discrete-time and continuous-time. In more detail, the models on the right side (without H) can be considered as the corresponding underlying models on the left side (containing H). For example, in a discrete-time HMM the underlying stochastic process is a DTMC (see Section 3.1). The models at the bottom (containing R) can be obtained from the corresponding one on the top (without R) by assigning a reward to states and/or transitions. The models on the back side (without D) are called probabilistic models which means that the successor of a state can be selected probabilistically. On the contrary, in the models on the front side (containing D) the probabilistic and nondeterministic choices coexist. A distribution over actions should be selected nondeterministically for a state, and then the successor can be chosen probabilistically according to the selected distribution over actions.

Many authors studied these models and concentrated on defining different logics to specify properties over some certain models. We start with the discrete-time models.

The temporal logic Probabilistic CTL (PCTL) [20, 2, 1, 23] is a probabilistic extension of CTL [18, 26, 12] where the probabilistic operator is introduced. PCTL can be interpreted over DTMCs (see



Figure 1.1: The Relations of the models

Chapter 2). Hansson & Jonsson [20] presented model checking algorithms for PCTL over DTMCs. The logic PCTL* [23, 1, 3] is a combination of PCTL and QLS (quantitative LTL specification) [3]. Iyer & Narasimha [23] described how to model check PCTL* over DTMCs.

Andova *et al.* [2] defined the logic Probabilistic Reward CTL (PRCTL) which extends PCTL by some operators to specify constraints over rewards. They presented a PRCTL model checking algorithm over discrete-time Markov Reward Models (DMRMs) where the reward is assigned only to states, not to transitions.

On the one hand, a Markov Decision Process (MDP) is a generalization of an MM in which probabilistic and nondeterministic choices coexist. On the other hand, an MDP can be viewed as a special case of an MDRP where the reward function is not considered. Puterman [32] presented the model discrete-time Markov Decision Processes (MDRPs) in more detail, for instance, the finite-horizon MDRPs, infinite-horizon MDRPs and discounted MDRPs. Note that Puterman called MDP what we call MDRP here, and the rewards there are assigned to both states and transitions. The logic PCTL* can be extended to interpret properties over MDPs using *schedulers* [17, 3]. De Alfaro [17] presented a PCTL* model checking algorithm over MDPs and Baier [17] adapted it over MDPs with fairness assumptions.

In this paper we will present the logic Probabilistic Observation CTL* (POCTL*) which can be viewed as an extension of the logic PCTL* where the next operator is equipped with an observation. POCTL* can be interpreted over the discrete-time HMM. We will extend the PCTL* model checking algorithm by Iyer & Narasimha [23] to a POCTL* model checking algorithm in Chapter 5 (Model Checking).

The Hidden (or Partially Observable) Markov Decision Reward Process (HMDRP or POMDRP) [35, 42, 21, 31, 28] is an orthogonal combination of HMM, MRM and MDP. Given an HMDRP, the usual objective function is to construct a control policy to maximize an objective value (e.g., the discounted cumulative reward). Until now, there is no logic which can be interpreted over HMDRPs.

Now we come to the continuous-time models. With the temporal logic Continuous Stochastic

Logic (CSL) [7] one can specify properties over CTMCs. In CSL the next and until operator are equipped with a real time interval which allow us to specify real-time probabilistic properties on CTMCs. Baier *et al.* [7] presented an efficient model checking algorithm for CSL over CTMCs.

In paper [5], the continuous-time, stochastic reward-based logic (CSRL) was introduced. In CSRL, the CSL-like next and until operators are augmented with an additional reward-interval-bound, thus, CSL is naturally a sublogic of CSRL. By CSRL model checking over continuous-time MRMs, however, only the case was considered where the time and reward constraints intervals have the form $[0, t]$ with $t \in \mathbb{R}_+$.

For continuous-time Markov decision processes (CTMDPs), an efficient algorithm [6] is presented to compute the maximum (or minimum) probability to reach a set of goal states within a given time bound in a uniformized CTMDP, i.e., a CTMDP in which the delay time distribution per state visit is the same for all states.

Chapter 2

Preliminaries

In this chapter we first present some basic definitions. Then we recall the logic PCTL* and its sublogics. Next to that, we recall how to construct a Büchi automaton for a given LTL formula. This automaton will be used for LTL and QLS model checking. At the end of this chapter, we show that the PCTL* model checking algorithm can be reduced to the QLS model checking algorithm. The logic PCTL* will be extended to the logic POCTL* in Chapter 4. In Chapter 5 we will adapt it to produce a POCTL* model checking algorithm.

2.1 Some Definitions

Distribution: Let S be a nonempty finite set. A *distribution* μ on S is a function $\mu : S \rightarrow [0, 1]$ such that $\sum_{s \in S} \mu(s) = 1$. Let $Distr(S)$ denote the set of distributions on S .

σ -algebra: Let S be a set, $\mathcal{P}(S)$ be the power set of S . Then a σ -algebra \mathcal{F} is a nonempty subset of $\mathcal{P}(S)$ (i.e., $\mathcal{F} \subseteq \mathcal{P}(S)$) such that the following hold:

- $S \in \mathcal{F}$
- $A \in \mathcal{F}$ implies $S \setminus A \in \mathcal{F}$
- $A_i \in \mathcal{F} \forall i \in \mathbb{N}$ implies $\bigcup_{i \in \mathbb{N}} A_i \in \mathcal{F}$

Note the power set $\mathcal{P}(S)$ itself is a σ -algebra. Let A be a subset of $\mathcal{P}(S)$, trivially we can find at least one σ -algebra $\mathcal{F}(A)$ which contains A . By taking the intersection of all σ -algebras containing A , we obtain *the smallest σ -algebra*. We call the smallest σ -algebra containing A the σ -algebra generated by A . A set equipped with a σ -algebra of subsets is called a *measurable space*.

Measure: A measure is formally defined as a nonnegative function: $P : \mathcal{F} \rightarrow \mathbb{R}_+$ from a σ -algebra \mathcal{F} to the reals such that $P(\emptyset) = 0$, and if A_n is a countable sequence in \mathcal{F} and pairwise disjoint, then

$$P\left(\bigcup_{n \in \mathbb{N}} A_n\right) = \sum_{n \in \mathbb{N}} P(A_n) \quad (2.1)$$

If additionally $P(S) = 1$, we call P a probability measure on \mathcal{F} . A probability space is defined as a triple (S, \mathcal{F}, P) on the domain S where (S, \mathcal{F}) is a measurable space, \mathcal{F} is the set of measurable subsets of S , and P a probability measure on \mathcal{F} .

Kripke Structure: We denote the set of atomic propositions by AP . A Kripke structure is a triple (S, \mathbf{R}, L) where S is a finite, nonempty set of states, $\mathbf{R} \subseteq S \times S$ is a transition relation on S , and $L \in S \rightarrow \mathcal{P}(AP)$ is a labeling function which assigns to each state $s \in S$ the atomic propositions $L(s)$ that are valid in S .

Intuitively, S represents the possible states, function L indicates which atomic propositions are valid in a given state, and $(s, s') \in \mathbf{R}$ iff there is a transition from s to s' . Now we define paths given a Kripke structure:

Paths Let $\mathcal{M} = (S, \mathbf{R}, L)$ be a Kripke structure. An infinite path σ of \mathcal{M} is a sequence s_0, s_1, \dots such that $(s_i, s_{i+1}) \in \mathbf{R}$ for all $i \in \mathbb{N}$. A finite path σ is a sequence s_0, \dots, s_l such that $(s_i, s_{i+1}) \in \mathbf{R}$ for all $i < l$ and $(s_l, s) \notin \mathbf{R} \forall s \in S$.

We say that an infinite path has length ∞ and a finite path $\sigma = s_0, \dots, s_l$ has length l . To identify a certain position in a path, we use $\sigma_s[i]$ to get the $(i+1)$ st state of σ if the length of σ is bigger equal than i . We use $\sigma[i]$ to get the suffix path of σ starting with $\sigma_s[i]$, i.e., s_i, s_{i+1}, \dots . Let $Path^{\mathcal{M}}$ denote the set of paths (finite and infinite) in \mathcal{M} , and $Path^{\mathcal{M}}(s)$ be the set of paths in \mathcal{M} that start with s . If \mathcal{M} is clear from the context, we simply omit the superscript \mathcal{M} .

DTMC: We define the discrete-time Markov Chains (DTMCs) [22, 8, 2] by equipping the transition relations of a Kripke structure with probabilities. Formally, a labeled DTMC \mathcal{D} is a triple (S, \mathbf{P}, L) where S is a finite, nonempty set of states, $\mathbf{P} : S \times S \rightarrow [0, 1]$ is a probability function satisfying $\sum_{s' \in S} \mathbf{P}(s, s') \in \{0, 1\}$ for all $s \in S$, and $L : S \rightarrow \mathcal{P}(AP)$ is a function which assigns to each state $s \in S$ the atomic propositions $L(s)$ that are valid in S .

The interpretation of S and L are the same as in a Kripke structure. $\mathbf{P}(s, s') > 0$ iff there is a transition from s to s' , the value indicates the probability that this transition will be taken if the model resides in state s . In the sequel, we assume there is an order on the set of states (e.g., the order in which we enumerate the states in S), and we consider \mathbf{P} as a probability matrix. In a DTMC the state evolves only at integer time points (i.e., $t=0,1,2,\dots$). A state s is called absorbing iff $\sum_{s' \in S} \mathbf{P}(s, s') = 0$; otherwise, s is called stochastic¹. The notations for paths ($Path^{\mathcal{D}}, Path^{\mathcal{D}}(s)$) that were introduced for Kripke structure carry over to DTMC in the obvious way. The only differences are that we write $\mathbf{P}(s, s') > 0$ instead of $(s, s') \in \mathbf{R}$ and $\mathbf{P}(s, s') = 0$ instead of $(s, s') \notin \mathbf{R}$.

Example 2.1.1. We consider the DTMC $\mathcal{D} = (S, \mathbf{P}, L)$ depicted in Figure 2.1. The state set S equals $\{f, u_1, u_2\}$ which are represented by circles. The transition probability is defined by $\mathbf{P}(s, s) = 0.8$ and $\mathbf{P}(s, s') = 0.1$ for $s \neq s'$. We label state $s \in S$ with the atomic proposition at_s .

2.2 The Logic PCTL* and Its Sublogics

CTL*: CTL* [26, 9, 18, 12] is a branching-time temporal logic which allows to state properties about possible paths that start with a state. The most elementary expressions are atomic propositions, i.e., propositions that cannot be further refined. Let a be an atomic proposition. The syntax of CTL* is defined in Backus-Naur Form where we let Φ, Ψ range over state formulas and ϕ, ψ range over path formulas:

$$\begin{aligned} \Phi &:= a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathbf{E}\phi \\ \phi &:= \Phi \mid \neg\phi \mid \phi \wedge \phi \mid \mathbf{X}\phi \mid \phi \mathcal{U} \phi \end{aligned}$$

¹In a fully probabilistic system (FPS), an extension of DTMCs, it is possible that $\sum_{s' \in S} \mathbf{P}(s, s') \in (0, 1)$, such a state is called sub-stochastic.

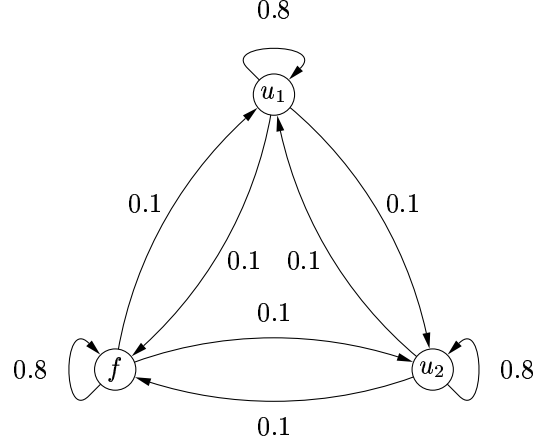


Figure 2.1: A DTMC

Other usual boolean operators are derived as: $\phi \vee \psi = \neg(\neg\phi \wedge \neg\psi)$, $tt = a \vee \neg a$, $ff = \neg tt$ and $\phi \Rightarrow \psi = \neg\phi \vee \psi$. The temporal operators \diamond (“eventually”) and \square (“always”) are defined by: $\diamond\phi = tt \mathcal{U} \phi$ and $\square\phi = \neg\diamond\neg\phi$. The formula $\mathbf{E}\phi$ is true in a state s iff there is a path which starts with s and satisfies ϕ . We write $\mathbf{A}\phi$ for $\neg\mathbf{E}\neg\phi$, i.e., ϕ is satisfied by all paths.

CTL* is a combination of the sublogics CTL (Computation Tree Logic) [18, 26, 12] and LTL (Propositional Linear Time Logic) [3, 23, 18, 12]. In CTL, every occurrence of a linear temporal operator \mathbf{X} (or \mathcal{U}) is immediately preceded by a path quantifier \mathbf{E} (or \mathbf{A}). Formally, CTL is the sublogic of CTL* whose state and path formulas are defined by:

$$\begin{aligned} \Phi &:= a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathbf{E}\phi \mid \mathbf{A}\phi \\ \phi &:= \mathbf{X}\Phi \mid \Phi \mathcal{U} \Phi \end{aligned}$$

The other sublogic LTL allows arbitrary combinations of path formulas but only propositional state formulas. Formally, LTL is defined by:

$$\phi := a \mid \neg\phi \mid \phi \wedge \phi \mid \mathbf{X}\phi \mid \phi \mathcal{U} \phi$$

The set of CTL* (and CTL) formulas consists of the set of state formulas, and is interpreted over branching models, e.g., over a Kripke structure. By comparison, the set of LTL formulas consists of the set of path formulas, and is interpreted over paths. To compare these logics there are different ways of interpreting LTL over branching structures. In this thesis we consider that an LTL formula ϕ holds for all paths, then we can illustrate the expressiveness of the three logics [26] as depicted in Figure 2.2.

The formula $\mathbf{A}(\diamond(a \wedge \mathbf{X}a))$ is an LTL formula and there is no equivalent CTL formula. On the contrary, the formula $\mathbf{A}\square\mathbf{E}\diamond a$ is a CTL formula and there is no equivalent LTL formula. Obviously, the conjunction of the two formulas is a CTL* formula but there is no equivalent CTL or LTL formula. Thus, CTL and LTL are not comparable, and both are proper sublogics of CTL*.

QLS [3]: LTL can also be interpreted over probabilistic models. Let $p \in [0, 1]$ and $\trianglelefteq \in \{\leq, <, \geq, >\}$. We call a pair $(\phi, \trianglelefteq p)$ a QLS (quantitative LTL specification) formula. Let $\mathcal{D} = (S, \mathbf{P}, L)$ be a DTMC with $s \in S$, we define

$$\mathcal{D}, s \models (\phi, \trianglelefteq p) \iff \Pr_s\{\sigma \in Path(s) \mid \sigma \models \phi\} \trianglelefteq p$$

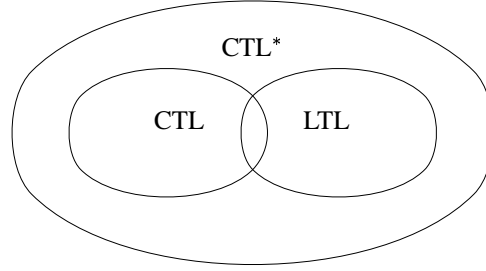


Figure 2.2: Expressiveness of the logics CTL*, CTL and LTL

where $\Pr_s\{\sigma \in Path(s) \mid \sigma \models \phi\}$ denotes the probability measure w.r.t. state s of all paths $\sigma \in Path(s)$ satisfying ϕ . The definition of the probability measure \Pr_s and the fact that the set $\{\sigma \in Path(s) \mid \sigma \models \phi\}$ is measurable will be shown later in Chapter 4 and Chapter 5 respectively.

Semantics of CTL*: Given a Kripke structure $\mathcal{M} = (S, \mathbf{R}, L)$ with $s \in S$, and $\sigma \in Path$, the semantics of CTL* is defined by a satisfaction relation (denoted by \models) either between a state s and a state formula Φ , or between a path σ and a path formula ϕ . We write $\mathcal{M}, s \models \Phi$ and $\mathcal{M}, \sigma \models \phi$ if state s , path σ satisfy state formula Φ , path formula ϕ respectively. If the model is clear from the context, we simply write $s \models \Phi$ and $\sigma \models \phi$. The relation \models is given in Figure 2.3.

$s \models a$	iff	$a \in L(s)$
$s \models \neg\Phi$	iff	$s \not\models \Phi$
$s \models \Phi \wedge \Psi$	iff	$s \models \Phi \wedge s \models \Psi$
$s \models \mathbf{E}\phi$	iff	$\exists \sigma \in Path(s). \sigma \models \phi$
$\sigma \models \Phi$	iff	$\sigma_s[0] \models \Phi$
$\sigma \models \neg\phi$	iff	$\sigma \not\models \phi$
$\sigma \models \phi \wedge \psi$	iff	$\sigma \models \phi \wedge \sigma \models \psi$
$\sigma \models \mathbf{X}\phi$	iff	$\sigma[1] \models \phi$
$\sigma \models \phi \mathcal{U} \psi$	iff	$\exists j \geq 0. (\sigma[j] \models \psi \wedge \forall 0 \leq i < j. \sigma[i] \models \phi)$

Figure 2.3: Semantics of CTL*

The interpretation of most constructs is straightforward. A state s satisfies $\mathbf{E}\phi$ if some path starting with the state s satisfies ϕ , while a path satisfies a state formula if the first state does. \mathbf{X} represents a *next-time operator* as usual, and $\phi \mathcal{U} \psi$ holds by a path if ϕ remains true until ψ becomes true. We can use the above definition to interpret the other derived boolean and temporal operators. For more detail, we refer to [18, 26, 9]. The formal semantics of CTL and LTL in a Kripke structure are intuitively clear from the interpretation of CTL*.

PCTL*: The logic PCTL* [23, 1, 3] expresses *quantitative* stochastic properties of systems which are themselves modeled as (discrete-time) Markov Models. It can be considered as a probabilistic variant of CTL* by adding the probabilistic operator \mathcal{P} . Formally, if ϕ is a PCTL* path formula (the same syntax as CTL* path formulas), then $\mathcal{P}_{\triangleleft p}(\phi)$ is a PCTL* state formula where $p \in [0, 1]$, and $\triangleleft \in \{\leq, <, \geq, >\}$.

PCTL* is the set of state formulas, i.e., the CTL* state formulas and the new probabilistic operator \mathcal{P} . The sublogic PCTL [20, 23, 2, 1] is a restricted version of PCTL*, just as CTL is a restricted version of CTL*. Its definition is extended from CTL by adding the operator \mathcal{P} , in a very similar way to the definition of PCTL*. Thus, the logic PCTL* can be also considered as a combination of PCTL and QLS.

Given a DTMC $\mathcal{D} = (S, \mathbf{P}, L)$, the semantics of PCTL* [1, 7, 23] is defined by a satisfaction relation between a state s (path σ) and a state formula Φ (path formula ϕ). We give only the semantics of the new operator \mathcal{P} :

$$s \models \mathcal{P}_{\leq p}(\phi) \quad \text{iff} \quad p_s(\phi) \leq p$$

where $p_s(\phi)$ denotes $\Pr_s\{\sigma \in \text{Path}(s) \mid \sigma \models \phi\}$. If ϕ is an LTL formula, $s \models \mathcal{P}_{\leq p}(\phi)$ is equivalent to $s \models (\phi, \leq p)$. We shall see later in Section 2.4 that PCTL* model checking problem boils down to corresponding QLS model checking problem.

Relationship of PCTL* and Its Sublogics: Figure 2.4 gives an overview of the relationship of PCTL* and its sublogics. There is an arrow from logic A to logic B if A is a proper sublogic of B. A logic in the upper part can be considered as the probabilistic counterpart to the corresponding one in the lower part.

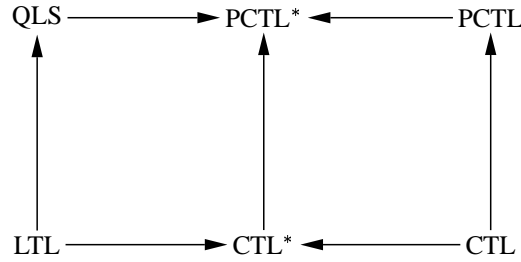


Figure 2.4: Relationship of the logic PCTL* and its sublogics

2.3 Constructing Büchi Automata from LTL Formulas

Vardi & Wolper [41, 38] showed that for any LTL-formula ϕ a Büchi automaton A_ϕ can be constructed on the alphabet $\mathcal{P}(AP)$ such that $\mathcal{L}_\omega(A_\phi)$, i.e., the language accepted by A_ϕ , equals the sequences of sets of atomic propositions satisfying ϕ . But the algorithm presented always yields the worst case result of $\mathcal{O}(2^{|\phi|})$ states. Gerth *et al.* [19, 16, 37] presented more efficient algorithms which work on-the-fly, however, the worst case complexity is the same. We briefly recall the results in [19].

First, we give the interpretation of the LTL formulas over infinite words [19, 40] over the alphabet $\mathcal{P}(AP)$. We assume that the given LTL formula does not contain bounded until formulas. In Section 5.3.4, we will deal with this restriction. We let w be an infinite sequence w_0, w_1, w_2, \dots over the alphabet $\mathcal{P}(AP)$. We write $w[i]$ for the suffix of w starting with w_i . The interpretation is given in Figure 2.5. Note that this interpretation is very similar to the one over paths, since we can consider w_i as the label of a state and w as a path in a Kripke structure (or DTMC) with $S \subseteq \mathcal{P}(AP)$.

The main idea of the construction is as follows. First, we push all negations inside, until they only precede atomic propositions. Second, a graph is constructed from the transformed formula. Then, we obtain a generalized Büchi automaton from the graph. Finally, the generalized Büchi automaton is transformed to a Büchi automaton.

$w \models a$	iff	$a \in w_0$ for $a \in AP$
$w \models \neg\phi$	iff	$w \not\models \phi$
$w \models \phi \wedge \psi$	iff	$w \models \phi \wedge w \models \psi$
$w \models \mathbf{X}\phi$	iff	$w[1] \models \phi$
$w \models \phi \mathcal{U} \psi$	iff	$\exists j \geq 0. (w[j] \models \psi \wedge \forall 0 \leq i < j. w[i] \models \phi)$

Figure 2.5: Interpretation of LTL over $\mathcal{P}(AP)$

2.3.1 Normal Form LTL Formulas

A normal form LTL formula [26] ϕ is an LTL formula where all negations in ϕ only precede atomic propositions. In order to deal with the negated until formula, we introduce the dual operator \mathcal{V} of \mathcal{U} by:

$$w \models \phi \mathcal{V} \psi \quad \text{iff} \quad \forall j \geq 0. (w[j] \models \psi \vee \exists 0 \leq i < j. w[i] \models \phi)$$

Intuitively, a word w satisfies $\phi \mathcal{V} \psi$, if either ψ holds infinitely often, or up to the point where ϕ releases the obligation. Now an LTL formula ϕ can be iteratively transformed into a normal form LTL formula using following rules:

$$\begin{aligned} \neg(\phi \vee \psi) &\longrightarrow (\neg\phi) \wedge (\neg\psi) & \neg(\phi \wedge \psi) &\longrightarrow (\neg\phi) \vee (\neg\psi) \\ \neg(\phi \mathcal{U} \psi) &\longrightarrow \neg\phi \mathcal{V} \neg\psi & \neg(\phi \mathcal{V} \psi) &\longrightarrow \neg\phi \mathcal{U} \neg\psi \\ \neg\mathbf{X}\phi &\longrightarrow \mathbf{X}(\neg\phi) \end{aligned}$$

2.3.2 Creating Graphs

The algorithm in Figure 2.6 builds a graph containing states and transitions. First, we introduce the structure of the graph nodes. A graph node contains fields *Name*, *Incoming*, *New*, *Old* and *Next*. For $Node = (Name, Incoming, New, Old, Next)$, we write $Name(Node) = Name$, $In(Node) = Incoming$, $New(Node) = New$, $Old(Node) = Old$ and $Next(Node) = Next$. *Name* is a string that is the name of the node. *Incoming* is the set of the names of the predecessors. A special element of *Incoming*, i.e., *init*, is used to mark initial nodes exclusively. *New* is the set of formulas that have not yet been processed, and *Old* is the set of the formulas that have already been processed. *Next* contains those formulas that must hold in all immediate successors of *Node*. The set *NodesSet* contains all graph nodes whose construction is finished, i.e., the new field is empty.

We let the line numbers in the following description refer to the algorithm in Figure 2.6. For a given normal form LTL formula ϕ_0 , the procedure $CREATGRAPH(\phi_0)$ gives a set of nodes whose constructions are finished (*NodesSet*). Actually, the procedure $EXPAND(Node, NodesSet)$ is called where *Node* equals $(NAME(), \{init\}, \{\phi_0\}, \emptyset, \emptyset)$ (line 28) and the set *NodesSet* is empty at the beginning. *Node* has a single incoming edge *init* to indicate that it is an initial node. It has initially only the obligation ϕ_0 in *New* and the sets *Old* and *Next* are initially empty.

The procedure $EXPAND(Node, NodesSet)$ checks whether there are unprocessed obligations left in *New* of *Node* (line 1). If not, *Node* is fully processed. If there has already been a node in *NodesSet* with the same obligations in both *Old* and *Next* fields (line 2), the node that already exists needs only to be updated w.r.t. its set of incoming edges (line 3). If no such node exists in *NodesSet*, *Node* is added to it, and a new node is created for its successor as described in lines 5-6.

If there are still obligations left in *New*, a formula η in *New* is removed from this set. In the case that η is a proposition or the negations of a proposition, either the current node is discarded (lines 12-13) or η is added to *Old* (lines 14-15). If η equals $\phi \wedge \psi$, both ϕ and ψ are added to *New* as the truth

```

EXPAND(Node, NodesSet)
1  if New(Node) =  $\emptyset$  then
2    if  $\exists N \in \text{NodesSet}$  with  $\text{Old}(N) = \text{Old}(Node)$  and  $\text{Next}(N) = \text{Next}(Node)$ 
3    then  $\text{In}(N) := \text{In}(N) \cup \text{In}(Node)$ ;
4      return (NodesSet);
5    else return (EXPAND((NAME(), Name(Node), Next(Node),  $\emptyset$ ,  $\emptyset$ ),
6      {Node}  $\cup$  NodesSet));
7  else
8    let  $\eta \in \text{New}(Node)$ ;
9     $\text{New}(Node) := \text{New}(Node) \setminus \{\eta\}$ ;
10   case  $\eta$  of
11      $\eta \in AP$  or  $\neg\eta \in AP$  or  $\eta = tt$  or  $\eta = ff \Rightarrow$ 
12       if  $\eta = ff$  or  $\neg\eta \in \text{Old}(Node)$  /* Current node contains a contradiction */
13       then return(NodesSet) /* Discard current node */
14       else  $\text{Old}(Node) := \text{Old}(Node) \cup \{\eta\}$ 
15         return (EXPAND(Node, NodesSet));
16      $\eta = \phi \wedge \psi \Rightarrow$ 
17       return (EXPAND((Name(Node), In(Node), New(Node)  $\cup$  ( $\{\phi, \psi\} \setminus \text{Old}(Node)$ ),
18         Old(Node)  $\cup$   $\{\eta\}$ , Next(Node)), NodesSet));
19      $\eta = \mathbf{X}\phi \Rightarrow$ 
20       return (EXPAND((Name(Node), In(Node), New(Node), Old(Node)  $\cup$   $\{\eta\}$ ,
21         Next(Node)  $\cup$   $\{\phi\}$ ), NodesSet));
22      $\eta = \phi \mathcal{U} \psi$  or  $\phi \mathcal{V} \psi$  or  $\phi \vee \psi \Rightarrow$ 
23       Node1 := (NAME(), In(Node), New(Node)  $\cup$  ( $\{\text{NEW1}(\eta)\} \setminus \text{Old}(node)$ ),
24         Old(Node)  $\cup$   $\{\eta\}$ , Next(Node)  $\cup$   $\{\text{NEXT1}(\eta)\}$ );
25       Node2 := (NAME(), In(Node), New(Node)  $\cup$  ( $\{\text{NEW2}(\eta)\} \setminus \text{Old}(node)$ ),
26         Old(Node)  $\cup$   $\{\eta\}$ , Next(Node));
27       return (EXPAND(Node2, EXPAND(Node1, NodesSet)));

CREATEGRAPH( $\phi$ )
28 return (EXPAND((NAME(), {init},  $\{\phi\}$ ,  $\emptyset$ ,  $\emptyset$ ),  $\emptyset$ ));

```

Figure 2.6: The algorithm for constructing a graph for an LTL formula

of both formula is needed to make η hold (lines 16–18). If η is a next-formula, say $\mathbf{X}\phi$, it suffices that ϕ holds at all immediate successors of $Node$. Thus, ϕ is added to $Next$ (lines 19–21). In the case that η is a disjunction, a \mathcal{U} - or a \mathcal{V} -formula, the current node is split into two nodes (lines 22–27) and new formulas can be added to the fields New and $Next$. The function $\text{NAME}()$ generates a new string for each call. The functions $\text{NEW1}(\eta)$, $\text{NEW2}(\eta)$ and $\text{NEXT1}(\eta)$ are defined in Table 2.1.

η	$\text{NEW1}(\eta)$	$\text{NEXT1}(\eta)$	$\text{NEW2}(\eta)$
$\phi \mathcal{U} \psi$	$\{\phi\}$	$\{\phi \mathcal{U} \psi\}$	$\{\psi\}$
$\phi \mathcal{V} \psi$	$\{\psi\}$	$\{\phi \mathcal{V} \psi\}$	$\{\phi, \psi\}$
$\phi \vee \psi$	$\{\phi\}$	\emptyset	$\{\psi\}$

Table 2.1: Splitting functions

For \mathcal{U} - and \mathcal{V} -formulas, the splitting is explained by observing that $\phi \mathcal{U} \psi$ is equivalent to $\psi \vee (\phi \wedge \mathbf{X}(\phi \mathcal{U} \psi))$, and $\phi \mathcal{V} \psi$ is equivalent to $\psi \wedge (\phi \vee \mathbf{X}(\phi \mathcal{V} \psi))$. For example, the upper node in Figure 2.7 is split into two nodes according to the algorithm.

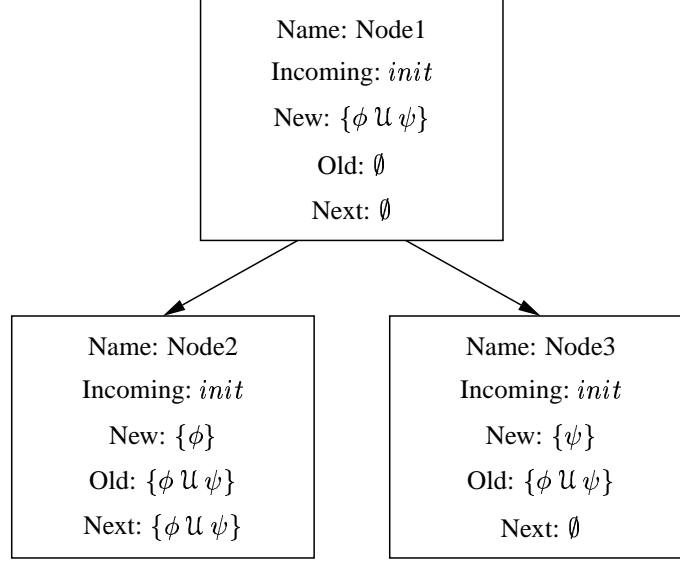


Figure 2.7: Splitting the until formula

Let $G = (V, E)$ be a graph where V is the nodes returned by the algorithm. If $p \in In(q)$, we define that there is a transition from node p to node q , i.e., $(p, q) \in E$.

Example 2.3.1. Let a and b be atomic propositions, and $\phi = a \mathcal{U} \mathbf{X}b$. Applying the algorithm $CREATEGRAPH(\phi)$, we obtain the graph depicted in Figure 2.8. Below each node q_i , O and N indicate $Old(q_i)$ and $Next(q_i)$, respectively. \square

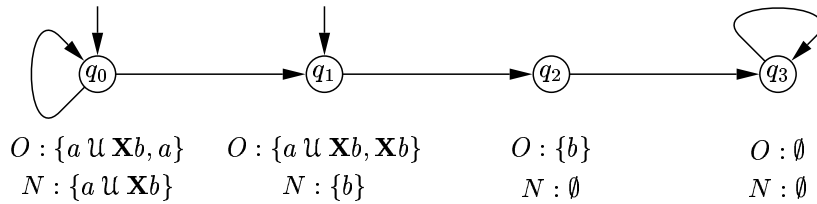


Figure 2.8: The graph for $a \mathcal{U} \mathbf{X}b$

2.3.3 The Generalized Büchi Automaton

A generalized Büchi automaton [19, 16, 40, 26] is a tuple $\mathcal{A} = (\Sigma, Q, L, \delta, Q_0, \mathcal{F})$ where Σ is an alphabet, Q is a set of states, $L : Q \rightarrow \mathcal{P}(\Sigma)$ is a labeling function, $\delta : Q \rightarrow \mathcal{P}(Q)$ is a transition function, $S_0 \subseteq S$ is a set of initial states and $\mathcal{F} \subseteq \mathcal{P}(S)$ is a set of accepting state sets.

A run π of \mathcal{A} is an infinite sequence $q_0, q_1, \dots \in Q^\omega$ such that $q_{i+1} \in \delta(q_i)$ for all $i \in \mathbb{N}$. A run $\pi = q_0, q_1, \dots$ is called an execution if additionally $q_0 \in Q_0$. Let $inf(\pi)$ be the set of states that

appear infinitely often in π . An execution π is accepting if $\text{inf}(\pi) \cap F \neq \emptyset$ for all $F \in \mathcal{F}$. Let $\pi[i]$ denote the suffix of the run π starting with q_i . An infinite word $w = w_0, w_1, \dots \in \Sigma^\omega$ is accepted by the automaton A if there is an accepting execution $\pi = q_0, q_1, \dots$ such that $w_i \in L(q_i)$. In this case, we also say that the execution π accepts the word w .

Let $G = (V, E)$ be the graph constructed as described in the last section. We define a generalized Büchi automaton $\mathcal{A}_\phi = (\Sigma, Q, L, \delta, Q_0, \mathcal{F})$ for the LTL formula ϕ as follows. The alphabet Σ is $\mathcal{P}(AP)$. The set of states Q equals V , i.e., the nodes set returned by the algorithm. The initial states Q_0 are those states q such that $\text{init} \in \text{In}(q)$. We have a transition $p \rightarrow q$ if $(p, q) \in E$, i.e., if $p \in \text{In}(q)$. The label of a state q is all sets in $\mathcal{P}(AP)$ that are compatible with $\text{Old}(q)$. More precisely, let $\text{Pos}(q)$ be $\text{Old}(q) \cap AP$ and $\text{Neg}(q)$ be $\{\eta \in AP \mid \neg\eta \in \text{Old}(q)\}$, i.e., $\text{Pos}(q)$ and $\text{Neg}(q)$ are the positive and negative occurrences of atomic propositions in q , respectively. Then, the label of state q is defined by:

$$L(q) = \{X \mid X \subseteq AP \wedge \text{Pos}(q) \subseteq X \wedge X \cap \text{Neg}(q) = \emptyset\}$$

For each subformula of ϕ of the type $\psi_1 \cup \psi_2$, we define a set $F \in \mathcal{F}$ which includes the states $q \in Q$ such that either $\psi_1 \cup \psi_2 \notin \text{Old}(q)$, or $\psi_2 \in \text{Old}(q)$. The construction of acceptance sets avoids accepting a run q_0, q_1, \dots in which $\psi_1 \cup \psi_2$ appears from some node q_i onwards without ψ_2 occurring later. Obviously, we have $|\mathcal{F}| < |\phi|$. The correctness of the construction follows from [19] (see Appendix B):

Theorem 2.3.2. [19] *Let Σ denote the alphabet $\mathcal{P}(AP)$. The generalized Büchi automaton obtained by transforming the graph as described above accepts exactly those infinite words over Σ that satisfy the LTL formula ϕ .*

Example 2.3.3. The Büchi automaton \mathcal{A}_ϕ for the LTL formula $\phi = a \cup \mathbf{X}b$ is depicted in Figure 2.9. The states are represented by circles. $L(q)$ contains all subsets of AP which are compatible with the atomic propositions near the state q . For example, we have $L(q_0) = \{\{a\}, \{a, b\}\}$, $L(q_1) = \mathcal{P}(AP)$. The initial states can be identified by an incoming arrow, i.e., $Q_0 = \{q_0, q_1\}$. The final states are marked with a double circle, i.e., $\mathcal{F} = \{\{q_1, q_2, q_3\}\}$.

We consider the word $w = \{a, b\}, \{a\}, \$, \{b\}, \$, \$ \dots$ where $\$$ represents arbitrary subsets of atomic propositions. We use the notation w_i and π_i to denote the i -th term of w and π respectively. w is accepted by \mathcal{A}_ϕ since we have the corresponding execution $\pi = q_0, q_0, q_1, q_2, q_3, q_3, \dots$ such that $w_i \in L(\pi_i)$ and $\text{inf}(\phi) = \{q_3\}$ contains a final state. In contrast, the word $w' = \{a\}, \{a\}, \{a\}, \{a\}, \{a\} \dots$ is not accepted. \square

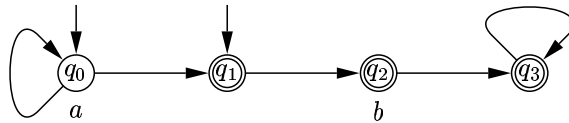


Figure 2.9: The generalized Büchi automaton for $a \cup \mathbf{X}b$

2.3.4 The Büchi Automaton

A Büchi automaton [3, 26, 23, 40] \mathcal{A} is a tuple $(\Sigma, Q, L, \delta, Q_0, F)$ where all components are the same as for a generalized Büchi automaton, except that $F \subseteq Q$ is a set of accepting states.

Büchi automata differ from Generalized Büchi automata by their acceptance condition. For a Büchi automaton \mathcal{A} , the requirement is that some state of the set F appears infinitely often, i.e., $\text{inf}(\pi) \cap F \neq \emptyset$. The definitions of run, execution, accepting execution that were introduced for generalized Büchi automata carry over to Büchi automata in the obvious way.

Lemma 2.3.4. [26, 40] *Given a generalized Büchi automaton, one can construct an equivalent Büchi automaton.*

Proof. Let $\mathcal{A} = (\Sigma, Q, L, \delta, Q_0, \mathcal{F})$ be a generalized Büchi automaton with $\mathcal{F} = \{F_1, \dots, F_k\}$. The equivalent Büchi automaton $\mathcal{A}' = (\Sigma, Q', L', \delta', Q'_0, F)$ defined as follows accepts the same language as \mathcal{A} :

- $Q' = Q \times \{i \mid 0 < i \leq k\}$
- $L'((q, i)) = L(q)$
- $(q', i) \in \delta'((q, i))$ iff $q' \in \delta(q)$ and $q \notin F_i$
 $(q', (i \bmod k) + 1) \in \delta'((q, i))$ iff $q' \in \delta(q)$ and $q \in F_i$
- $Q'_0 = Q_0 \times \{1\}$
- $F = F_1 \times \{1\}$ □

Actually, we can choose an arbitrary $0 < i \leq k$ for the initial and accepting states of \mathcal{A}' . Thus, the automaton \mathcal{A}' is not uniquely determined. The idea of the construction is as follows. In \mathcal{A}' we have k levels of states, and we go from state (q, i) in level i to state $(q', (i \bmod k) + 1)$ in level $(i \bmod k) + 1$ if q is in F_i . If a run of \mathcal{A}' visits some final state $(q, 1)$ infinitely often, it has to repeatedly go through all levels and back to state $(q, 1)$. Thus, all sets in \mathcal{F} are visited infinitely often. The preceding lemma concludes the description of how to obtain a Büchi automaton \mathcal{A}_ϕ for a given LTL-formula ϕ . We recall the complexity of the construction:

Theorem 2.3.5. [26, 38] *A Büchi automaton can be constructed for an LTL formula ϕ with complexity $\mathcal{O}(k \cdot 2^{|\phi|})$, where k is the number of the acceptance sets in the corresponding generalized Büchi automaton, and $k < |\phi|$.* □

2.4 Model Checking for PCTL* and Its Sublogics

LTL Model checking: Given a Kripke structure $\mathcal{M} = (S, \mathbf{R}, L)$ with $s \in S$ and an LTL formula ϕ , the LTL model checking problem [38, 4, 26, 29] is to determine whether $\mathcal{M}, s \models \phi$. Pnueli & Manna [29] gave an LTL model checking algorithm based on the construction of a temporal tableau of the LTL formula ϕ . Vardi & Wolper [41, 38] obtained a different algorithm based on Büchi automata (ω -automata). The complexities of the two model checking algorithms are the same. We recall briefly the second one.

Vardi & Wolper [38] proposed the following model checking algorithm: First, we construct a Büchi automaton \mathcal{A}_ϕ for the desired LTL formula ϕ and a Büchi automaton $\mathcal{A}_\mathcal{M}$ for the Kripke structure \mathcal{M} with $s_0 \in S^2$. Then, we check whether all sequences³ accepted by the automaton $\mathcal{A}_\mathcal{M}$ satisfy the formula ϕ . We know that the automaton \mathcal{A}_ϕ accepts exactly the sequences satisfying the formula ϕ , thus the model checking problem reduces to the automata-theoretic problem of checking that $\mathcal{L}_\omega(\mathcal{A}_\mathcal{M}) \subseteq \mathcal{L}_\omega(\mathcal{A}_\phi)$, i.e., all sequences accepted by the automaton $\mathcal{A}_\mathcal{M}$ are also accepted by the automaton \mathcal{A}_ϕ . Equivalently, we check whether $\mathcal{L}_\omega(\mathcal{A}_\mathcal{M}) \cap \mathcal{L}_\omega(\overline{\mathcal{A}_\phi}) = \emptyset$ where $\mathcal{L}_\omega(\overline{\mathcal{A}_\phi}) = \mathcal{L}_\omega(\mathcal{A}_{\neg\phi})$. Note this is much more efficient than to check the inclusion of Büchi automata. Let $|\mathcal{M}| = |S| + |\mathbf{R}|$, we recall the complexity of LTL model checking:

Theorem 2.4.1. [38] **Time and Space Complexity of LTL Model Checking**

Checking whether an LTL formula ϕ is satisfied by a final state Kripke structure \mathcal{M} can be done in time $\mathcal{O}(|\mathcal{M}| \cdot 2^{|\phi|})$ and in space $\mathcal{O}((\log|\mathcal{M}| + |\phi|)^2)$. □

²Given $\mathcal{M} = (S, \mathbf{R}, L)$ and a state $s_0 \in S$, one can construct the Büchi automaton $\mathcal{A}_\mathcal{M} = (\Sigma, S, L, \delta, \{s_0\}, S)$ where $\Sigma = \mathcal{P}(AP)$ and $s' = \delta(s)$ iff $(s, s') \in R$.

³Recall that a state s satisfies ϕ if ϕ holds for all paths starting with s .

QLS Model Checking: Given a DTMC $\mathcal{D} = (S, \mathbf{P}, L)$ and a QLS formula $(\phi, \trianglelefteq p)$, the model checking problem [23, 14, 15] for QLS is to check whether $\mathcal{D}, s \models (\phi, \trianglelefteq p)$. We briefly recall the algorithm presented by Iyer & Narasimha [23].

We view automata and DTMCs as directed graphs. A strongly connected component (or SCC for short) of a graph is a maximal subgraph in which every node can reach every other node of the subgraph. A bottom SCC (BSCC) of a graph is an SCC such that every state that is reachable from a state of the SCC is in the SCC.

Let $q_{init} \notin Q$. Let $\mathcal{A}_\phi = (\Sigma, Q, L, \delta, Q_0, F)$ be the constructed Büchi automaton for ϕ . We define an equivalent transition labeled Büchi automaton⁴ $\mathcal{A}'_\phi = (\Sigma, Q', \delta', \{q_{init}\}, F)$, where $Q' = Q \cup \{q_{init}\}$, and $\delta' : Q' \times \mathcal{P}(\Sigma) \rightarrow Q'$ with $q_0 \in \delta'(q_{init}, L(q_0))$ iff $q_0 \in Q_0$, and $q' \in \delta'(q, L(q'))$ iff $q' \in \delta(q)$.

The main idea of the model checking algorithm is as follows. First, we construct a transition labeled Büchi automaton \mathcal{A}'_ϕ for the given LTL formula ϕ . Then, the product automaton $\mathcal{D} \times \mathcal{A}'_\phi$ is constructed. Finally, the QLS model checking problem will be reduced to a reach probability analysis. The following lemma characterizes the SCCs⁵ of the product automaton which contribute to the measure of paths that satisfy ϕ .

Theorem 2.4.2. [23] *Let $\mathcal{D} \times \mathcal{A}'_\phi$ be the product of the DTMC \mathcal{D} and the Büchi automaton \mathcal{A}'_ϕ with a unique initial state q_{init} . The measure of the set $\{\sigma \in \text{Path}^\mathcal{D}(s) \mid \sigma \models \phi\}$ is equal to the measure of the set of projections onto \mathcal{D} of paths in $\text{Path}^{\mathcal{D} \times \mathcal{A}'_\phi}((s, q_{init}))$ which loop in an SCC containing a final state.*

Based on the preceding lemma, Iyer & Narasimha presented an algorithm to check whether $\mathcal{D}, s \models (\phi, \trianglelefteq p)$. This will be used for a PCTL* model checking algorithm. Later we will adapt it to QOS model checking (Chapter 5) which will be used for a POCTL* model checking algorithm.

CTL Model Checking: Given a Kripke structure $\mathcal{M} = (S, \mathbf{R}, L)$, a state s and a CTL formula Φ , the model checking problem for CTL [26, 13] is to check whether the formula Φ is valid in state s . As shown in Figure 2.10, the basic concept of the model checking algorithm is to compute the set of states which satisfy the formula Φ and then to check whether s is an element of this set.

Let $\text{Sat}(\Phi) = \{s \in S \mid s \models \Phi\}$ denote the set of states which satisfy Φ . We compute the set $\text{Sat}(\Phi)$ recursively as follows. The cases where Φ is of the form $tt, a, \neg\Phi', \Phi_1 \wedge \Phi_2$ can be derived easily by:

$$\begin{aligned} \text{Sat}(tt) &= S & \text{Sat}(\neg\Phi') &= S \setminus \text{Sat}(\Phi') \\ \text{Sat}(a) &= \{s \in S \mid a \in L(s)\} & \text{Sat}(\Phi_1 \wedge \Phi_2) &= \text{Sat}(\Phi_1) \cap \text{Sat}(\Phi_2) \end{aligned}$$

The set $\text{Sat}(\mathbf{EX}\Phi)$ contains all the states that can reach some state in $\text{Sat}(\Phi)$ by a single transition, formally:

$$\text{Sat}(\mathbf{EX}\Phi) = \{s \in S \mid \exists(s, s') \in \mathbf{R} \wedge s' \in \text{Sat}(\Phi)\}$$

The computation of $\text{Sat}(\mathbf{E}(\Phi \cup \Psi))$ and $\text{Sat}(\mathbf{A}(\Phi \cup \Psi))$ is based on the concept of the *fix-points* as the one presented by Katoen [26]. We associate each formula ϕ with the set $\text{Sat}(\phi)$. Let $F : \mathcal{P}(S) \rightarrow \mathcal{P}(S)$ be a function where S is a finite set. F is called *monotonic* if for $S_1, S_2 \in \mathcal{P}(S)$ with $S_1 \subseteq S_2$ we have $F(S_1) \subseteq F(S_2)$. We call $S_1 \in \mathcal{P}$ a *fix-point* of F if $F(S_1) = S_1$, and S_1 is called the *least fix-point* if additionally for any fix-point $S_2 \in \mathcal{P}(S)$ we have $S_1 \subseteq S_2$. The *greatest fix-point* of F is defined similarly. We cite the following theorem according to Katoen:

⁴Iyer & Narasimha assumed that the Büchi automaton is transition labeled, and contains a unique initial state q_{init} .

⁵In [23] Iyer & Narasimha considered the BSCCs instead of SCCs in the product automaton. But to prove the correctness, they used a Lemma which says that if a BSCC B in the DTMC is the projection of an SCC B' in the product automaton, B' is a BSCC. This is wrong. Appendix C contains a counterexample.

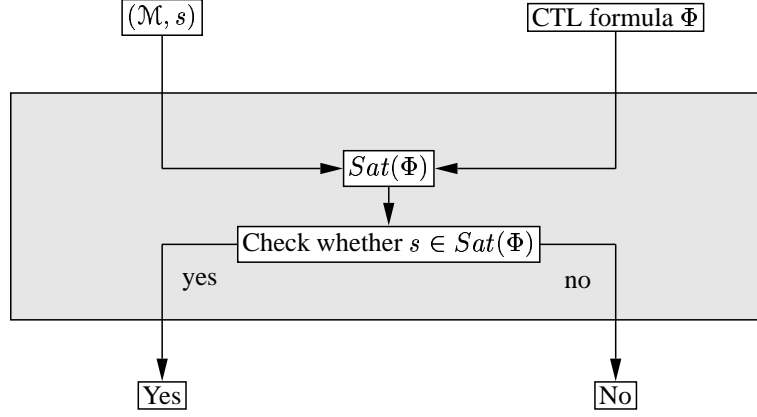


Figure 2.10: CTL Model checking

Theorem 2.4.3. [26] **Fix-point characterizations of CTL-formulas**

1. $Sat(\mathbf{E}(\Phi \cup \Psi))$ is the least fix-point of $F(Z) = Sat(\Psi) \cup (Sat(\Phi) \cap \{s \in S \mid \exists s' \in R(s) \cap Z\})$
2. $Sat(\mathbf{A}(\Phi \cup \Psi))$ is the least fix-point of $F(Z) = Sat(\Psi) \cup (Sat(\Phi) \cup \{s \in S \mid \forall s' \in R(s) \cap Z\})$

Then, the problems of computing the sets $Sat(\mathbf{E}(\Phi \cup \Psi))$ and $Sat(\mathbf{A}(\Phi \cup \Psi))$ boil down to computing the least fix-point of corresponding F . According to Tarski-Knaster theorem⁶ this can be computed by means of iterations $\emptyset, F(\emptyset), F^2(\emptyset), \dots$. Since S is a finite set, this procedure is guaranteed to terminate, i.e., there exists some $k' \leq |S|$ such that $F^{k'+1}(\emptyset) = F^{k'}(\emptyset)$.

Based on the preceding theorem, Katoen presented algorithms to obtain the sets $Sat(\mathbf{E}(\Phi \cup \Psi))$ and $Sat(\mathbf{A}(\Phi \cup \Psi))$ with complexity $\mathcal{O}(|\phi| \times |\mathcal{M}|^3)$ where $|\mathcal{M}| = |S| + |\mathbf{R}|$. Clarke *et al.* [13] presented a more efficient algorithm with time complexity $\mathcal{O}(|\phi| \times |\mathcal{M}|^2)$.

Recall that LTL model checking is exponential in the size of the formula. The difference in time complexity with respect to the length of the formula seems drastic, but the formulas in LTL are always shorter, and (in the worst case) even exponentially shorter than their equivalent formulation in CTL (e.g., the Hamilton path problem [26]).

CTL* Model Checking: Given a Kripke structure $\mathcal{M} = (S, \mathbf{R}, L)$, a state s and a CTL* state formula Φ , the model checking problem for CTL* is to check whether the formula Φ is valid in state s . This can be reduced to LTL model checking as described by Emerson & Lei:

Theorem 2.4.4. [18] *If we are given an algorithm LMC to solve the LTL model checking problem, we can design an algorithm BMC to solve the branching-time model checking problem (in particular, CTL*) – which subsumes the linear logic in expressive power – of the same order of complexity as LMC.*

Proof. The cases where Φ is of the form $a, \neg\Phi', \Phi_1 \wedge \Phi_2$ are identical to the CTL model checking algorithm. If $\Phi = \mathbf{E}\phi$, where ϕ is an arbitrary path formula, we let $\mathbf{E}\phi_1, \dots, \mathbf{E}\phi_k$ be the list of all “top level” proper \mathbf{E} -subformulas of ϕ . If this list is empty, we simply run the LMC for ϕ . Otherwise, we recursively model check each $\mathbf{E}\phi_i$ and obtain the set $Sat(\mathbf{E}\phi_i)$ where $i = 1, \dots, k$. Then, we

⁶Let S be a set with n states. If $F : \mathcal{P}(S) \rightarrow \mathcal{P}(S)$ is a monotonic function, then $F^n(\emptyset)$ is the least fix-point of F and $F^n(S)$ is the greatest fix-point of F .

replace the subformulas $\mathbf{E}\phi_1, \dots, \mathbf{E}\phi_k$ by new atomic propositions a_1, \dots, a_k and extend the label of state s by a_i iff $s \in \text{Sat}(\mathbf{E}\Phi_i)$. Let ϕ' be the path formula resulting from substituting each a_i for its corresponding $\mathbf{E}\phi_i$ in ϕ . Call the linear model checker LMC for ϕ' . When it returns each state is labeled with $\mathbf{E}\phi'$ or $\neg\mathbf{E}\phi'$. Replacing back the a_i by $\mathbf{E}\phi_i$ we get each state labeled with $\mathbf{E}\phi$ or $\neg\mathbf{E}\phi$ appropriately. \square

PCTL Model Checking: Let $\mathcal{D} = (S, \mathbf{P}, L)$ be a DTMC with $s \in S$, and Φ be a PCTL formula. The algorithm to check whether $\mathcal{D}, s \models \Phi$ is based on the CTL model checking algorithm. Since the logic PCTL is obtained by adding the operators \mathcal{P} to CTL, we only need to consider the case that Φ is of the form $\mathcal{P}_{\leq p}(\phi)$ where ϕ is a PCTL path formula. We briefly recall the results by Hansson & Jonsson [20, 3] where the until formula is equipped with a superscript to specify the maximal number of steps (see Section 4.1 for the definition of the bounded until formula).

We compute $p_s(\phi) = \Pr_s\{\sigma \in \text{Path}(s) \mid \sigma \models \phi\}$ for all state $s \in S$ and then we put $\text{Sat}(\Phi) = \{s \in S \mid p_s(\phi) \leq p\}$. If ϕ is the next operator, we have: $p_s(\mathbf{X}\Psi) = \sum_{s' \in \text{Sat}(\Psi)} \mathbf{P}(s, s')$. To compute $p_s(\Phi \mathcal{U}^{\leq n} \Psi)$ we discuss first $n < \infty$. We let $p_s(\Phi \mathcal{U}^{\leq n} \Psi)$ equal 0 for $n < 0$, equal 1 if $s \in \text{Sat}(\Psi)$, and equal 0 if $s \in S \setminus (\text{Sat}(\Phi) \cup \text{Sat}(\Psi))$. Otherwise we have

$$p_s(\Phi \mathcal{U}^{\leq n} \Psi) = \sum_{s' \in S} \mathbf{P}(s, s') \cdot p_{s'}(\Phi \mathcal{U}^{\leq n-1} \Psi)$$

The algorithm based on this equation has complexity $\mathcal{O}(t \times |S|^2)$. We can formulate this equation in terms of matrix multiplication.

Lemma 2.4.5. [20] Let $S_i = \text{Sat}(\Phi) \setminus \text{Sat}(\Psi)$, i.e., the set of states which are labeled with Φ but not Ψ . We define the $|S| \times |S|$ -matrix M by

$$M(s, s') = \begin{cases} \mathbf{P}(s, s') & \text{if } s \in S_i \\ 1 & \text{if } s = s' \wedge s \notin S_i \\ 0 & \text{otherwise.} \end{cases}$$

We let $\mathbf{p}(\Phi \mathcal{U}^{\leq n} \Psi)$ denote a column vector over states with $\mathbf{p}(\Phi \mathcal{U}^{\leq n} \Psi)(s) = p_s(\Phi \mathcal{U}^{\leq n} \Psi)$. Then,

$$\mathbf{p}(\Phi \mathcal{U}^{\leq n} \Psi) = M^n \times \mathbf{p}(\Phi \mathcal{U}^{\leq 0} \Psi)$$

Proof. By definition of the vector \mathbf{p} , it is sufficient to show

$$\begin{aligned} p_s(\Phi \mathcal{U}^{\leq n} \Psi) &= (M^n \times \mathbf{p}(\Phi \mathcal{U}^{\leq 0} \Psi))(s) \\ &= (M \times \mathbf{p}(\Phi \mathcal{U}^{\leq n-1} \Psi))(s) = M(s) \times \mathbf{p}(\Phi \mathcal{U}^{\leq n-1} \Psi) \end{aligned}$$

for all state $s \in S$ where $M(s)$ denotes the row vector of M w.r.t. state s . We consider three cases:

1. If $s \in \text{Sat}(\Psi)$ then (since $s \notin S_i$) $M(s, s')$ equals 1 if $s = s'$ and equals 0 otherwise. Thus, $p_s(\Phi \mathcal{U}^{\leq n} \Psi) = p_s(\Phi \mathcal{U}^{\leq n-1} \Psi)$. Since $p_s(\Phi \mathcal{U}^{\leq 0} \Psi) = 1$, we conclude that $p_s(\Phi \mathcal{U}^{\leq n} \Psi) = 1$.
2. If $s \notin \text{Sat}(\Phi) \cup \text{Sat}(\Psi)$ then (similar to case 1) we obtain that $p_s(\Phi \mathcal{U}^{\leq n} \Psi) = 0$.
3. If $s \in S_i$ then $M(s, s') = \mathbf{P}(s, s')$. Thus, $p_s(\Phi \mathcal{U}^{\leq n} \Psi) = \sum_{s' \in S} \mathbf{P}(s, s') \cdot p_{s'}(\Phi \mathcal{U}^{\leq n-1} \Psi)$.

As a result, $\mathbf{p}(\Phi \mathcal{U}^{\leq n} \Psi)$ can be recursively calculated by $M^n \times \mathbf{p}(\Phi \mathcal{U}^{\leq 0} \Psi)$. \square

The algorithm based on the previous lemma has complexity $\mathcal{O}(\log(t) \times |S|^3)$. In case $n = \infty$, we define a set of failure states by: $S^{NO} = \{s \in S \mid p_s(\Phi \mathcal{U} \Psi) = 0\}$ and a set of success states

by: $S^{YES} = \{s \in S \mid p_s(\Phi \cup \Psi) = 1\}$. (These two sets can be easily identified [20, 15].) Then, $p_s(\Phi \cup \Psi)$ equals 1 if $s \in S^{YES}$, equals 0 if $s \in S^{NO}$ and otherwise:

$$p_s(\Phi \cup \Psi) = \sum_{s' \in S} \mathbf{P}(s, s') \cdot p_{s'}(\Phi \cup \Psi)$$

These equations can be solved with a complexity of $\mathcal{O}((|S| - |S^{YES}| - |S^{NO}|)^{2.81})$. Hansson & Jonsson also presented efficient algorithms ($\mathcal{O}(S)$) for the case $p = 0$ and $p = 1$.

PCTL* Model Checking: Given a DTMC $\mathcal{D} = (S, \mathbf{P}, L)$ with $s \in S$ and a PCTL* formula Φ , the PCTL* model checking problem [23, 15, 14, 3] is to check whether $\mathcal{D}, s \models \Phi$. This can be reduced to the QLS model checking problem. In more detail, we deal with the operators $a, \neg\Phi, \Phi \wedge \Psi$ using the same techniques proposed for CTL* and CTL. We only need to consider the case that Φ is of the form $\mathcal{P}_{\leq p}(\phi)$ where ϕ is a PCTL* path formula. Let Φ_1, \dots, Φ_k be the maximal state subformulas of ϕ , then for $i = 1, \dots, k$, the set $Sat(\Phi_i)$ can be obtained recursively. We replace the subformulas Φ_1, \dots, Φ_k by new atomic propositions a_1, \dots, a_k and extend the label of state s by a_i iff $s \in Sat(\Phi_i)$. Let ϕ' be the path formula resulting from substituting each a_i for its corresponding Φ_i in ϕ , then we only need to apply the QLS model checking algorithm to check whether $s \models (\phi', \leq p)$.

Chapter 3

Hidden Markov Models

This chapter first recalls the concept of HMM, then presents the induced DTMC, paths over HMM, and probability spaces for a given HMM.

3.1 Labeled Discrete-Time HMMs

An HMM [33, 34] is a doubly embedded stochastic process with an underlying stochastic process that is *hidden*, but can only be observed through another set of stochastic processes that produce a sequence of observations. We add a labeling function to the standard definition of HMMs, in other words, we consider an HMM as an extension of a DTMC:

Definition 3.1.1. Labeled Discrete-Time HMMs

A labeled discrete-time HMM \mathcal{H} is a tuple $(S, \mathbf{P}, L, \Theta, \mu, \alpha)$ where (S, \mathbf{P}, L) is a labeled DTMC, Θ is a finite set of observations, $\mu : S \times \Theta \rightarrow [0, 1]$ is an observation function satisfying $\mu(s, \cdot) \in \text{Distr}(\Theta) \forall s \in S$, and $\alpha \in \text{Distr}(S)$ is an initial distribution on S . \square

Intuitively, \mathcal{D} is the underlying DTMC, and the observation set Θ corresponds to the output of the model. By definition, $\mu(s, \cdot)$ is a distribution on Θ , and $\mu(s, o)$ indicates the probability that the state s produces the observation o . Recall that a state of a DTMC evolves only at integer time points. For the sake of brevity, we write $\mu_s(o)$ instead of $\mu(s, o)$. The probability that the model starts with state s is $\alpha(s)$. In what follows we use the term HMM to refer to a discrete-time HMM.

For technical reasons, we assume there is no absorbing state in an HMM throughout our discussion in the remainder of this thesis. As indicated by Baier [3] (for concurrent probabilistic system), this is a harmless restriction since any system can be transformed into an “equivalent” system without absorbing states. Given an HMM $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ with absorbing states, we insert a special state \dagger with a self-loop and transitions from any absorbing state in \mathcal{H} to \dagger . Formally, we define the set of atomic proposition by $AP' = AP \dot{\cup} \{at_\dagger\}$ and consider the system $\mathcal{H}' = (S', \mathcal{P}', L', \Theta', \mu', \alpha')$. The set S' equals $S \dot{\cup} \{\dagger\}$. The transition probability $\mathcal{P}'(\dagger, \dagger)$ equals 1, $\mathcal{P}'(s, \dagger)$ equals 1 if s is an absorbing state in \mathcal{H} , and $\mathcal{P}'(s, s')$ equals $\mathcal{P}(s, s')$ if $s, s' \in S$. The label of state \dagger is $\{at_\dagger\}$ and otherwise $L'(s) = L(s)$. The observation set Θ' equals $\Theta \dot{\cup} \{\dagger\}$. The observation function μ'_s equals μ_s if $s \in S$, and $\mu'_\dagger(\dagger) = 1$, i.e., the state \dagger is fully observable. Finally, α' is defined by $\alpha'(\dagger) = 0$, and $\alpha'(s) = \alpha(s)$ otherwise. Hence, we may assume w.l.o.g. that the system does not have absorbing states¹.

¹In the Section 4.2 (Semantics of POCTL*) we will perform an equivalent transformation from a formula over an HMM \mathcal{H} which contains absorbing states into a corresponding one over \mathcal{H}' .

We can view a DTMC as a special case of a discrete-time HMM, i.e., we let the observation set equal the state set (i.e., $S = \Theta$) and we can only observe s from itself (i.e., $\mu_s(s) = 1$). In this case, the model is nothing more than the underlying DTMC \mathcal{D} with an initial distribution α . Generally, in contrast, Θ differs from S , and the model provides us with only observations. The observation depends stochastically and exclusively on the current state. In general, the same observation could be emitted by several different states; therefore, we are uncertain about the current state.

Example 3.1.2. We consider a coin toss model taken from [33]. Assume that a dealer in a casino tosses coins. He may use a fair or one of two biased coins, and the probability that he changes a coin is only 0.1. We model it by an HMM $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ depicted in Figure 3.1. The state set S equals $\{f, u_1, u_2\}$ which are represented by circles where f is a fair coin and u_1, u_2 are two biased coins. We label state $s \in S$ with the atomic proposition at_s . The observation set Θ contains *head* and *tail*. The observation function μ is indicated near the state, e.g., $\mu_{u_1}(h) = 0.8$, which means that the biased coin u_1 produces the observation *head* with probability 0.8. The transition probabilities is defined by $P(s, s) = 0.8$ and $P(s, s') = 0.1$ for $s \neq s'$. The initial distribution is given by $\alpha(f) = \alpha(u_1) = \alpha(u_2) = \frac{1}{3}$. Actually, the underlying DTMC of \mathcal{H} is the one described in Example 2.1.1. \square

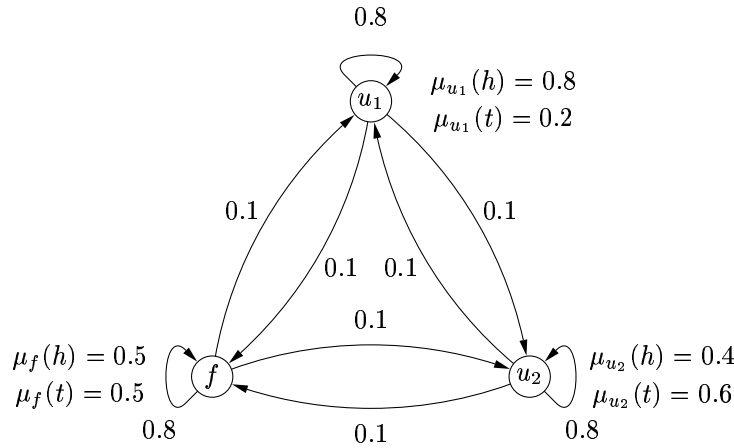


Figure 3.1: Coin Toss Model with one fair coin and two biased coins

3.2 Belief State

In an HMM, we are uncertain about the current state, but, we can summarize the historical observations in a *belief state* (or *information state*) [35, 42, 21, 31] which is a distribution over S . A belief state is not really a state of the HMM. Rather, it is a way to describe what we know about the state, given the history of observations. The set of all possible belief states, i.e., $\text{Distr}(S)$, is called the *belief space* as well, and is denoted by \mathcal{B} . We use S^t with $S^t \in S$ to denote the state at time t , and O^t with $O^t \in \Theta$ to denote the observation at time t . We write B^t to denote the belief state at time t .

Definition 3.2.1. Belief State

Let $o_i \in \Theta$ where $t = 0, \dots, t$. The belief state at time t , $B^t = b_t$, is the distribution over S at time t given the observation history o_0, \dots, o_t :

$$b_t(s) = P(S^t = s | O^0 = o_0, \dots, O^t = o_t, \mathcal{H}) \quad \forall s \in S \quad \square$$

To simplify the notations, we write o^t instead of $O^t = o$, then the belief state at time t becomes:

$$b_t(s) = P(s^t | o_0^0, \dots, o_t^t, \mathcal{H})$$

Now given the historical observations o_0, \dots, o_t , the question is how to calculate the belief state b_t . The belief state at time 0 only depends on the initial distribution and the first observation. The belief state at time t captures all of our information about the past. As a result, we can inductively calculate the current belief state b_t based on the previous belief state b_{t-1} and the current observation o_t . This is illustrated in Figure 3.2.

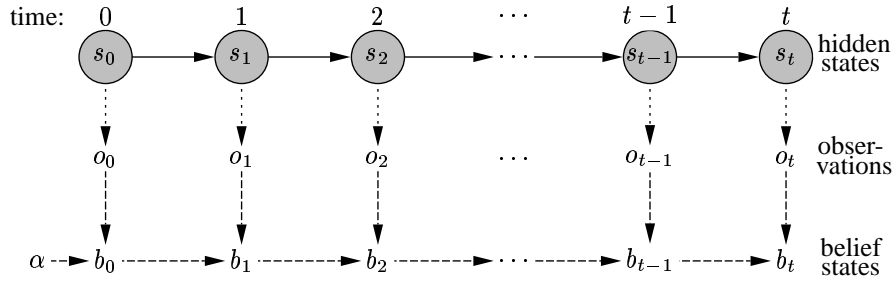


Figure 3.2: Updating belief states

We depict the states in the gray circles to indicate that they are hidden. The states together with the solid arrows between them represent the underlying state evolution. The dotted arrows between states and observations mean that the observation o_t is produced from the state s_t according to the observation function μ . The dashed arrows, between the current observation o_t , previous belief state b_{t-1} and the current belief state b_t , mean that b_t depends on o_t and b_{t-1} for all $t = 1, \dots, n$. As a particular case, b_0 is a deterministic function of o_0 and the initial distribution α . Recall the probability that the path starts with s_0 is given by $\alpha(s_0)$. Similar to states and observations, we write b^t instead of $B^t = b$.

Lemma 3.2.2. Update Function of Belief States

The belief state at time point 0 is:

$$b_0(s) = \frac{\alpha(s)\mu_s(o_0)}{K_0} \quad (3.1)$$

where K_0 is a normalizing constant with value $P(o_0^0 | \mathcal{H}) = \sum_{s \in S} \alpha(s)\mu_s(o_0)$. Now assume that we are given a belief state b_t and an observation o_{t+1} at time $t \geq 0$. The new belief state b_{t+1} is:

$$b_{t+1}(s) = \frac{\sum_{s_t \in S} b_t(s_t) \mathbf{P}(s_t, s) \mu_s(o_{t+1})}{K_{t+1}} \quad (3.2)$$

where K_{t+1} is a normalizing constant with value:

$$P(o_{t+1}^{t+1} | b_t^t, \mathcal{H}) = \sum_{s \in S} \left(\sum_{s_t \in S} b_t(s_t) \mathbf{P}(s_t, s) \mu_s(o_{t+1}) \right)$$

Proof. The belief state at time 0:

$$\begin{aligned}
b_0(s) &= P(s^0 | o_0^0, \mathcal{H}) && \text{(Definition of belief state)} \\
&= \frac{P(o_0^0, s^0 | \mathcal{H})}{P(o_0^0 | \mathcal{H})} && \text{(Bayesian Rule)} \\
&= \frac{P(s^0 | \mathcal{H}) P(o_0^0 | s^0, \mathcal{H})}{\sum_{s_0 \in S} P(o_0^0, s_0^0 | \mathcal{H})} = \frac{\alpha(s) \mu_s(o_0)}{K_0} && \text{(Chain Rule)}
\end{aligned}$$

Assume that we are given b_t and o_{t+1} :

$$\begin{aligned}
b_{t+1}(s) &= P(s^{t+1} | o_0^0, \dots, o_{t+1}^{t+1}, \mathcal{H}) && \text{(Definition of belief state)} \\
&= \frac{P(o_{t+1}^{t+1}, s^{t+1} | o_0^0, \dots, o_t^t, \mathcal{H})}{P(o_{t+1}^{t+1} | o_0^0, \dots, o_t^t, \mathcal{H})} && \text{(Bayesian Rule)} \\
&= \frac{P(o_{t+1}^{t+1}, s^{t+1} | o_0^0, \dots, o_t^t, \mathcal{H})}{\sum_{s_{t+1} \in S} P(o_{t+1}^{t+1}, s_{t+1}^{t+1} | o_0^0, \dots, o_t^t, \mathcal{H})} && \text{(Chain Rule)}
\end{aligned}$$

Using the chain rule on just the summand in the numerator, we get

$$P(o_{t+1}^{t+1}, s^{t+1} | o_0^0, \dots, o_t^t, \mathcal{H}) = P(s^{t+1} | o_0^0, \dots, o_t^t, \mathcal{H}) \cdot P(o_{t+1}^{t+1} | o_0^0, \dots, o_t^t, s^{t+1}, \mathcal{H})$$

The last term equals $P(o_{t+1}^{t+1} | s^{t+1}, \mathcal{H}) = \mu_s(o_{t+1})$ because the observations o_0, \dots, o_t are irrelevant given that $S^{t+1} = s$. The first term can be simplified using the chain rule:

$$\begin{aligned}
P(s^{t+1} | o_0^0, \dots, o_t^t, \mathcal{H}) &= \sum_{s_t \in S} P(s_t^t | o_0^0, \dots, o_t^t, \mathcal{H}) \cdot P(s^{t+1} | o_0^0, \dots, o_t^t, s_t^t, \mathcal{H}) \\
&= \sum_{s_t \in S} b_t(s_t) P(s^{t+1} | s_t^t, \mathcal{H}) = \sum_{s_t \in S} b_t(s_t) \mathbf{P}(s_t, s_{t+1})
\end{aligned}$$

Putting all this together we get Equation 3.2. \square

Intuitively, the probability to be in s at time $t+1$ equals $\sum_{s' \in S} b_t(s') \mathbf{P}(s', s)$, i.e., the one without taking the observation into account, weighted with $\mu_s(o_{t+1})$, the likelihood of observing o_{t+1} from state s . If we consider α, b_t as a row vector², we can rewrite equations 3.1 and 3.2 in matrix notation:

$$b_0 = \frac{\alpha D_{o_0}}{K_0}, b_{t+1} = \frac{b_t \mathbf{P} D_{o_{t+1}}}{K_{t+1}}$$

where D_o is a diagonal observation matrix with $D_o(s, s) = \mu_s(o)$.

Example 3.2.3. We consider the HMM in *Example 3.1.2* and assume that we get a first observation *head*. Then,

$$\alpha D_{head} = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3} \right) \begin{pmatrix} 0.5 & 0 & 0 \\ 0 & 0.8 & 0 \\ 0 & 0 & 0.4 \end{pmatrix} = \left(\frac{1}{6}, \frac{4}{15}, \frac{2}{15} \right)$$

therefore $K_0 = \frac{17}{30}$. After normalizing we get $b_0 = (0.294, 0.471, 0.235)$. If there follows another observation *tail*, then,

$$\begin{aligned}
b_0 \mathbf{P} D_{tail} &= (0.294, 0.471, 0.235) \begin{pmatrix} 0.8 & 0.1 & 0.1 \\ 0.1 & 0.8 & 0.1 \\ 0.1 & 0.1 & 0.8 \end{pmatrix} \begin{pmatrix} 0.5 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.6 \end{pmatrix} \\
&= (0.153, 0.086, 0.159)
\end{aligned}$$

²Recall given $S = \{u, v\}$, we consider (u, v) as an order on the set S . Hence, we can define the row vector $b_0 = (b_0(u), b_0(v))$ and the matrix \mathbf{P} by $\mathbf{P}_{00} = \mathbf{P}(u, u)$, $\mathbf{P}_{01} = \mathbf{P}(u, v)$, $\mathbf{P}_{10} = \mathbf{P}(v, u)$, $\mathbf{P}_{11} = \mathbf{P}(v, v)$

therefore $K_1 = 0.398$. After normalizing we get $b_1 = (0.384, 0.216, 0.400)$. \square

To simplify the notation, we define the *update functions* $\tau_0, \tau : \mathcal{B} \times \Theta \rightarrow \mathcal{B} \cup \{0\}$, which map from the product of belief space and observation set back to the belief space, as follows. For $t = 0$, we define $\tau_0(\alpha, o_0) = b_0$ if $P(o^0 | \alpha) > 0$ and $\tau_0(\alpha, o_0) = 0$ if $P(o^0 | \alpha) = 0$. For $t > 0$, we have similarly $\tau(b_{t-1}, o_t) = b_t$ if $P(o^t | b^{t-1}) > 0$ and $\tau(b_{t-1}, o_t) = 0$ if $P(o^t | b^{t-1}) = 0$.

Forward Variable [33, 34]: Now, we introduce the forward variable³ and give the relation to the belief state. The forward variable $\alpha_t(s)$ is defined as:

$$\alpha_t(s) = P(o_0^0, \dots, o_t^t, s^t | \mathcal{H})$$

i.e., the probability of the observation sequence o_0, \dots, o_t and of ending up in state s at time t . Using the chain rule, we obtain:

$$P(o_0^0, \dots, o_t^t | \mathcal{H}) = \sum_{s \in S} P(o_0^0, \dots, o_t^t, s^t | \mathcal{H}) = \sum_{s \in S} \alpha_t(s)$$

Applying an analogous proof to the one in Lemma 3.2.2, we can calculate the forward variable at time t recursively in a similar way to the one for belief state:

$$\alpha_0(s) = \alpha(s) \mu_s(o_0) \quad (3.3)$$

$$\alpha_{t+1}(s) = \sum_{s_t \in S} \alpha_t(s_t) \mathbf{P}(s_t, s) \mu_s(o_{t+1}) \quad (3.4)$$

Rewriting equations 3.3 and 3.4 in matrix notation:

$$\alpha_0 = \alpha D_{o_0}, b_{t+1} = b_t \mathbf{P} D_{o_{t+1}} \quad (3.5)$$

where D_o is a diagonal observation matrix with $D_o(s, s) = \mu_s(o)$.

Lemma 3.2.4. The Relation of Belief State and Forward Variable

We let $o_i \in \Theta$ and $K_i = P(o_i^i | b_{i-1}^{i-1}, \mathcal{H})$, i.e., the normalizing constant at time i . Then:

$$b_t = \frac{\alpha_t}{\prod_{i=0}^t K_i} \quad (3.6)$$

$$P(o_0^0, \dots, o_t^t | \mathcal{H}) = \sum_{s \in S} \alpha_t(s) = \prod_{i=0}^t K_i \quad (3.7)$$

Proof. We show Equation 3.6 by induction on t .

Basis of induction $t = 0$:

$$b_0(s) \stackrel{Eq. 3.1}{=} \frac{\alpha(s) \mu_s(o_0)}{K_0} \stackrel{Eq. 3.3}{=} \frac{\alpha_0(s)}{K_0}$$

Induction step $n \Rightarrow n + 1$:

$$b_{t+1}(s) = \frac{\sum_{s' \in S} b_t(s') \mathbf{P}(s', s) \mu_s(o_{t+1})}{K_{t+1}} \quad (\text{Equation 3.2})$$

$$= \frac{\sum_{s' \in S} \alpha_t(s') \mathbf{P}(s', s) \mu_s(o_{t+1})}{\left(\prod_{i=0}^t K_i\right) K_{t+1}} \quad (\text{Induction hypothesis})$$

$$= \frac{\alpha_{t+1}(s)}{\prod_{i=0}^{t+1} K_i} \quad (\text{Equation 3.4})$$

³This will be used later in the Section 3.4 (Paths in HMM and Probability Space on Paths). We introduce it here since the updating functions of the forward variable are very similar to the one of the belief state.

Equation 3.7: The first part is already shown, we prove the second part:

$$\sum_{s \in S} \alpha_t(s) \stackrel{Eq. 3.6}{=} \sum_{s \in S} \left(b_t(s) \prod_{i=0}^t K_i \right) = \prod_{i=0}^t K_i \left(\sum_{s \in S} b_t(s) \right) = \prod_{i=0}^t K_i \quad \square$$

3.3 Induced DTMC

An HMM $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ induces a stochastic process. As indicated in Figure 3.2 an arbitrary observation sequence o_0, \dots, o_n induces a belief state sequence b_0, \dots, b_n . The belief states in this sequence and the initial distribution α will be considered as state set of the induced process. This sequence also determines the transition relations between the belief states. What we should additionally pay attention to is that the belief state b_0 has a different update function as we have seen in Lemma 3.2.2.

Definition and Lemma 3.3.1. Induced DTMC

Let $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ be an HMM. Let \mathcal{B} denote the belief space and \perp represent the initial state with $\perp \notin \mathcal{B}$. The induced stochastic process is a DTMC⁴ $\mathcal{H}_D = (S_D, \mathbf{P}_D)$ where the state set $S_D \subset \{\perp\} \cup \mathcal{B}$ is defined inductively as follows. \perp is an element of S_D . For $o \in \Theta$, we let $\tau_0(\alpha, o) \in S_D$ if $\tau_0(\alpha, o) \neq 0$. For $b \in S_D \cap \mathcal{B}$, we let $\tau(b, o) \in S_D$ if $\tau(b, o) \neq 0$. The probability matrix $\mathbf{P}_D : S_D \times S_D \rightarrow \mathbb{R}$ is defined by $\mathbf{P}_D(\perp, \tau_0(\alpha, o)) = P(o^0 | \alpha, \mathcal{H})$ and $\mathbf{P}_D(b, \tau(b, o)) = P(o^{t+1} | b^t, \mathcal{H})$.

Proof. What we need to prove is that the state b in S_D is stochastic. We first assume $b \neq \perp$:

$$\begin{aligned} \sum_{b' \in S_D} \mathbf{P}_D(b, b') &= \sum_{o \in \Theta} \mathbf{P}_D(b, \tau(b, o)) = \sum_{o \in \Theta} \mathbf{P}(o^{t+1} | b^t) \\ &= \sum_{o \in \Theta} \sum_{s \in S} \sum_{s' \in S} b(s') \mathbf{P}(s', s) \mu_s(o) \\ &= \sum_{s \in S} \sum_{s' \in S} b(s') \mathbf{P}(s', s) \left(\sum_{o \in \Theta} \mu_s(o) \right) \\ &= \sum_{s' \in S} b(s') \left(\sum_{s \in S} \mathbf{P}(s', s) \right) = 1 \end{aligned}$$

The case that $b = \perp$ can be proven in a similar way. □

Example 3.3.2. We consider the HMM $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ depicted in Figure 3.3. The observation set Θ equals $\{A, B\}$ and the observation function is indicated near the state, e.g., $\mu_u(A) = \frac{2}{3}$, $\mu_u(B) = \frac{1}{3}$. The initial distribution is given by $\alpha(u) = \alpha(v) = \frac{1}{2}$.

The induced DTMC up to time 1 is depicted in Figure 3.4. It starts with state \perp which represents the initial distribution α . The labels of the arrows have the form $p(o)$ where $p \in (0, 1]$ and $o \in \Theta$. The transition from b to b' with label $p(o)$ means $b' = \tau(b, o)$ and $p = P(o|b)$ is the probability that the next observation is o given the current belief state b . We observe that it equals the corresponding normalizing constant K . □

By induction on t , we can easily show that the number of states at time t is bounded by $|\Theta|^{t+1}$. As a result the total number of states up to time n ,

$$\sum_{t=0}^{n+1} |\Theta|^t = \frac{|\Theta|^{n+2} - 1}{|\Theta| - 1}$$

⁴The labeling function is omitted.

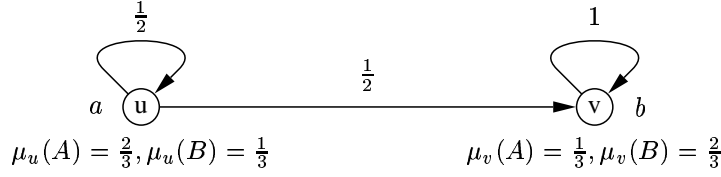


Figure 3.3: A simple HMM

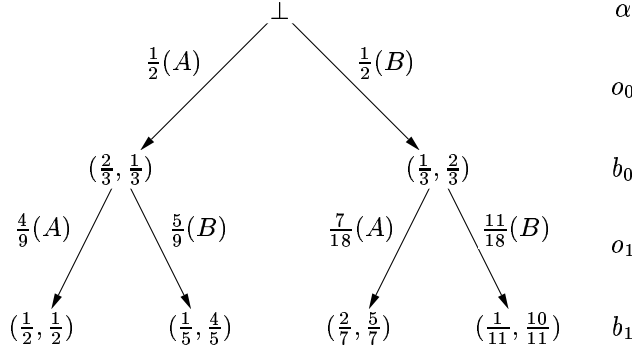


Figure 3.4: The induced DTMC of the HMM of Example 3.1.2 up to time 1

grows at most exponentially in the time.

3.4 Paths in HMM and Probability Spaces over Paths

Definition 3.4.1. Paths over \mathcal{H}

Given an HMM $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ with $s_i \in S$ and $o_i \in \Theta$ for all $i \in \mathbb{N}$. A path⁵ σ of \mathcal{H} is a sequence $(s_0, o_0) \rightarrow (s_1, o_1) \rightarrow \dots \in (S \times \Theta)^\omega$ where $\mu_{s_i}(o_i) > 0$, $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \in \mathbb{N}$ and $(S \times \Theta)^\omega$ denotes the set of infinite sequences of elements of $S \times \Theta$. \square

We introduce some useful notions. For a path σ and $i \in \mathbb{N}$, let $\sigma_s[i] = s_i$ denote the $(i+1)$ st state of σ , and $\sigma_o[i] = o_i$ denote the $(i+1)$ st observation of σ . Let $\sigma[i]$ denote the suffix path of σ starting with $\sigma_s[i]$, i.e., $(s_i, o_i) \rightarrow (s_{i+1}, o_{i+1}) \rightarrow \dots$. Note that $\sigma[0] = \sigma$.

Let $Path^{\mathcal{H}}$ denote the set of paths. If \mathcal{H} is clear from the context, we simply write $Path$ instead of $Path^{\mathcal{H}}$. Similar to CTMC [7], we define a probability space on paths of \mathcal{H} . We let I_n denote the index set $\{0, 1, \dots, n\}$, and we define the *basic cylinder set* as follows:

$$\mathcal{C}((s_0, o_0), (s_1, o_1), \dots, (s_n, o_n)) := \{\sigma \in Path \mid \forall i \in I_n. \sigma_s[i] = s_i \wedge \sigma_o[i] = o_i\}$$

It consists of all paths σ starting with $(s_0, o_0) \rightarrow (s_1, o_1) \rightarrow \dots (s_n, o_n)$. Note that it is possible that $\mu_{s_i}(o_i) = 0$ for some $i = 0, \dots, n$, and in this case the basic cylinder set is an empty set.

Lemma 3.4.2. Property of the Basic Cylinder Set

Let $C_1 = \mathcal{C}((s_0, o_0), \dots, (s_n, o_n))$ and $C_2 = \mathcal{C}((s'_0, o'_0), \dots, (s'_m, o'_m))$ be two basic cylinder sets, then

$$C_1 \cap C_2 = \emptyset \text{ or } C_1 \subseteq C_2 \text{ or } C_2 \subseteq C_1$$

⁵Actually, what we define here is an infinite path of \mathcal{H} . A finite path σ is a sequence $(s_0, o_0) \rightarrow (s_1, o_1) \rightarrow \dots (s_l, o_l)$ such that s_l is an absorbing state, $\mu_{s_i}(o_i) > 0$ for all $i \leq l$ and $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i < l$. As we have assumed that there are no absorbing states in \mathcal{H} , we consider only infinite paths.

If additionally $m = n$,

$$C_1 \cap C_2 = \emptyset \text{ or } C_2 = C_1$$

Proof. We first assume $n \geq m$, and we show respectively:

$$C_1 \cap C_2 = \emptyset \text{ or } C_1 \subseteq C_2$$

Let $\sigma \in C_1$, therefore σ starts with $(s_0, o_0), \dots, (s_n, o_n)$. If we have $s_i = s'_i$ and $o_i = o'_i$ for all $i \leq m$, then, $\sigma \in C_2$ and we obtain $C_1 \subseteq C_2$. Otherwise, $\sigma \notin C_2$ and therefore $C_1 \cap C_2 = \emptyset$. For the case $n \leq m$, we get symmetrically:

$$C_1 \cap C_2 = \emptyset \text{ or } C_2 \subseteq C_1 \quad \square$$

Let \mathcal{C} contain all sets $\mathcal{C}((s_0, o_0), \dots, (s_n, o_n))$ where s_0, \dots, s_n range over all state sequences and o_0, \dots, o_n range over all observation sequences. Let \mathcal{F} be the σ -algebra on *Path* generated by \mathcal{C} . Let $\mathbf{i}(s, s_0) = 1$ if $s = s_0$, and $\mathbf{i}(s, s_0) = 0$ if $s \neq s_0$. The probability measure⁶ \Pr_s on \mathcal{F} is defined by induction on n by $\Pr_s(\mathcal{C}(s_0, o_0)) = \mathbf{i}(s, s_0)\mu_{s_0}(o_0)$ and, for $n > 0$:

$$\begin{aligned} \Pr_s(\mathcal{C}((s_0, o_0), \dots, (s_n, o_n))) \\ = \Pr_s(\mathcal{C}((s_0, o_0), \dots, (s_{n-1}, o_{n-1}))) \cdot \mathbf{P}(s_{n-1}, s_n)\mu_{s_n}(o_n) \end{aligned}$$

By induction on n , we obtain:

$$\Pr_s(\mathcal{C}((s_0, o_0), \dots, (s_n, o_n))) = \mathbf{i}(s, s_0)\mu_{s_0}(o_0) \prod_{i=1}^n \mathbf{P}(s_{i-1}, s_i)\mu_{s_i}(o_i) \quad (3.8)$$

We define the cylinder sets over states and observations by:

$$\begin{aligned} \mathcal{C}(s_0, \dots, s_n) &:= \bigcup_{o_0, \dots, o_n \in \Theta} \mathcal{C}((s_0, o_0), (s_1, o_1), \dots, (s_n, o_n)) \\ \mathcal{C}(o_0, \dots, o_n) &:= \bigcup_{s_0, \dots, s_n \in S} \mathcal{C}((s_0, o_0), (s_1, o_1), \dots, (s_n, o_n)) \end{aligned}$$

By Equation 2.1 and Lemma 3.4.2, we have:

$$\Pr_s \left(\bigcup_{o_0, \dots, o_n \in \Theta} \mathcal{C}((s_0, o_0), \dots, (s_n, o_n)) \right) = \sum_{o_0, \dots, o_n \in \Theta} \Pr_s(\mathcal{C}((s_0, o_0), \dots, (s_n, o_n))) \quad (3.9)$$

There may exist some empty basic cylinder sets, but this doesn't affect our result. For the cylinder set $\mathcal{C}(o_0, \dots, o_n)$, we write $s_0, \dots, s_n \in S$ instead of $o_0, \dots, o_n \in \Theta$ under the \bigcup and \sum symbols in Equation 3.9.

Lemma 3.4.3. Probability Measure of $\mathcal{C}(s_0, \dots, s_n)$

Let $s, s_0, \dots, s_n \in S$. The probability measure w.r.t. state s , i.e., \Pr_s , of the cylinder set $\mathcal{C}(s_0, \dots, s_n)$ is given by induction on n by $\Pr_s(\mathcal{C}(s_0)) = \mathbf{i}(s, s_0)$ and, for $n > 0$:

$$\Pr_s(\mathcal{C}(s_0, \dots, s_n)) = \Pr_s(\mathcal{C}(s_0, \dots, s_{n-1})) \cdot \mathbf{P}(s_{n-1}, s_n)$$

⁶We define here actually a probability function \Pr_s on the set \mathcal{C} . For \mathcal{F} is a σ -algebra generated by \mathcal{C} , this probability function can be extended to a unique probability measure on \mathcal{F} .

Proof. It is sufficient to show $\Pr_s(\mathcal{C}(s_0, \dots, s_n)) = \mathbf{i}(s, s_0) \prod_{i=1}^n \mathbf{P}(s_{i-1}, s_i)$:

$$\Pr_s(\mathcal{C}(s_0, \dots, s_n)) = \sum_{o_0, \dots, o_n \in \Theta} \Pr_s(\mathcal{C}((s_0, o_0), \dots, (s_n, o_n))) \quad (\text{Equation 3.9})$$

$$\begin{aligned} &= \sum_{o_0, \dots, o_n \in \Theta} \left(\mathbf{i}(s, s_0) \mu_{s_0}(o_0) \prod_{i=1}^n \mathbf{P}(s_{i-1}, s_i) \mu_{s_i}(o_i) \right) \quad (\text{Equation 3.8}) \\ &= \mathbf{i}(s, s_0) \prod_{i=1}^n \mathbf{P}(s_{i-1}, s_i) \cdot \sum_{o_0, \dots, o_n \in \Theta} \left(\prod_{i=0}^n \mu_{s_i}(o_i) \right) \end{aligned}$$

The last term can be simplified by:

$$\sum_{o_0, \dots, o_n \in \Theta} \left(\prod_{i=0}^n \mu_{s_i}(o_i) \right) = \sum_{o_0 \in \Theta} \dots \sum_{o_n \in \Theta} \left(\prod_{i=0}^n \mu_{s_i}(o_i) \right) = \prod_{i=0}^n \left(\sum_{o_i \in \Theta} \mu_{s_i}(o_i) \right) = 1 \quad \square$$

In order to calculate the probability measure of the cylinder set $\mathcal{C}(o_0, \dots, o_n)$ we first expand the forward variable:

Lemma 3.4.4. Expansion of the Forward Variable

Let the forward variable be as defined by equations 3.3 and 3.4, then,

$$\sum_{s_n \in S} \alpha_n(s_n) = \sum_{s_0, \dots, s_n \in S} \left(\alpha(s_0) \mu_{s_0}(o_0) \prod_{i=1}^n \mathbf{P}(s_{i-1}, s_i) \mu_{s_i}(o_i) \right)$$

where we assume that the product equals 1 if $n = 0$.

Proof. It is sufficient to show:

$$\alpha_n(s_n) = \sum_{s_0, \dots, s_{n-1} \in S} \left(\alpha(s_0) \mu_{s_0}(o_0) \prod_{i=1}^n \mathbf{P}(s_{i-1}, s_i) \mu_{s_i}(o_i) \right)$$

We prove it by induction on n .

Basis of induction $n = 0$:

$$\alpha_0(s_0) \stackrel{\text{Eq. 3.3}}{=} \alpha(s_0) \mu_{s_0}(o_0) = \alpha(s_0) \mu_{s_0}(o_0) \overbrace{\prod_{i=1}^0 \mathbf{P}(s_{i-1}, s_i) \mu_{s_i}(o_i)}{=1}$$

Induction step $n \Rightarrow n + 1$:

$$\begin{aligned} \alpha_{n+1}(s_{n+1}) &\stackrel{\text{Eq. 3.4}}{=} \sum_{s_n \in S} \alpha_n(s_n) \mathbf{P}(s_n, s_{n+1}) \mu_{s_{n+1}}(o_{n+1}) \\ &\stackrel{ih}{=} \sum_{s_n \in S} \left(\sum_{s_0, \dots, s_{n-1} \in S} \left(\alpha(s_0) \mu_{s_0}(o_0) \prod_{i=1}^n \mathbf{P}(s_{i-1}, s_i) \mu_{s_i}(o_i) \right) \right) \mathbf{P}(s_n, s_{n+1}) \mu_{s_{n+1}}(o_{n+1}) \\ &= \sum_{s_0, \dots, s_n \in S} \left(\alpha(s_0) \mu_{s_0}(o_0) \prod_{i=1}^{n+1} \mathbf{P}(s_{i-1}, s_i) \mu_{s_i}(o_i) \right) \end{aligned}$$

where *ih* means induction hypothesis. Summing this equation over all the states $s_{n+1} \in S$ concludes the proof. \square

Lemma 3.4.5. Probability Measure of $\mathcal{C}(o_0, \dots, o_n)$

Let $s \in S$ and $o_0, \dots, o_n \in \Theta$. The probability measure w.r.t. state s , i.e., \Pr_s , of the cylinder set $\mathcal{C}(o_0, \dots, o_n)$ is given by:

$$\Pr_s(\mathcal{C}(o_0, \dots, o_n)) = \sum_{s_0, \dots, s_n \in S} \mathbf{i}(s, s_0) \mu_{s_0}(o_0) \left(\prod_{i=1}^n \mathbf{P}(s_{i-1}, s_i) \mu_{s_i}(o_i) \right)$$

Summing this probability measure over all states, we obtain:

$$\sum_{s \in S} \alpha(s) \Pr_s(\mathcal{C}(o_0, \dots, o_n)) = \sum_{s_n \in S} \alpha_n(s_n) = P(o_0^0, \dots, o_n^t | \mathcal{H})$$

Proof. Replacing the observations o_0, \dots, o_n by states s_0, \dots, s_n in Equation 3.9 we obtain:

$$\Pr_s(\mathcal{C}(o_0, \dots, o_n)) = \sum_{s_0, \dots, s_n \in S} \Pr_s(\mathcal{C}((s_0, o_0), \dots, (s_n, o_n)))$$

Applying Equation 3.8, we get immediately the first equation. The second equation can also be directly obtained by applying Lemma 3.4.4 and Lemma 3.2.4. \square

Lemma 3.4.5 shows that the probability to observe an observation sequence o_0, \dots, o_n coincides with the sum of $\Pr_s(\mathcal{C}(o_0, \dots, o_n))$ over all states, i.e., the probability measure of the cylinder set $\mathcal{C}(o_0, \dots, o_n)$, weighted by the initial distribution. Now the following lemma is obvious:

Lemma 3.4.6. Probability Space

Let $s \in S$. The triple $(Path, \mathcal{F}, \Pr_s)$ on domain $Path$ is a probability space, where \mathcal{F} is the σ -algebra generated by the set of basic cylinder sets \mathcal{C} , and \Pr_s is the probability measure which is defined by Equation 3.8. \square

Let $b \in \mathcal{B}$ be a belief state, and $C \in \mathcal{C}$ be a basic cylinder set. We extend the probability measure with respect to a belief state b by: $\Pr_b(C) = \sum_{s \in S} b(s) \cdot \Pr_s(C)$. Similar to Lemma 3.4.6, the triple $(Path, \mathcal{F}, \Pr_b)$ on domain $Path$ is also a probability space.

Chapter 4

The Logic POCTL*

This chapter presents the branching-time temporal logic Probabilistic Observation CTL* (POCTL*) which allows us to specify properties over HMMs. We have indicated in the introduction that for an HMM, one wants to specify properties over the underlying DTMC and in addition, one is also interested in reasoning about properties over the other set of stochastic processes which produce observations. The logic PCTL* is interpreted over DTMCs to express quantitative stochastic properties (see Section 2.2). We extend PCTL* to POCTL* such that the next operator is equipped with an observation constraint. In this way we can state properties over the observations, e.g., $X_o\phi$ means that the next observation is o and the subsequent path satisfies ϕ .

POCTL* can be also considered as a variant of the temporal logic ACTL* introduced by De Nicola *et al.* [30, 24]. ACTL* is interpreted over Labeled Transition Systems (LTS) and has been proven to have the same power as CTL*. In ACTL* the usual next operator is extended to interpret the labeled action of the transition (e.g., $X_a\phi$ means the next transition is labeled with an action a and the subsequent path satisfies ϕ).

4.1 Syntax of POCTL*

Definition 4.1.1. Syntax of POCTL*

Let $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ be an HMM with $o \in \Theta$. The syntax of the logic POCTL* is defined as follows:

$$\begin{aligned}\Phi &:= a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \epsilon \\ \phi &:= \Phi \mid \neg\phi \mid \phi \wedge \phi \mid \mathbf{X}_o\phi \mid \phi \mathcal{U}^{\leq n} \phi \\ \epsilon &:= \mathcal{P}_{\leq p}(\phi) \mid \neg\epsilon \mid \epsilon \wedge \epsilon\end{aligned}$$

where $n \in \mathbb{N}$ or $n = \infty$, $0 \leq p \leq 1$ and $\triangleleft \in \{\leq, <, \geq, >\}$. □

The syntax of POCTL* consists of state formula, path formula and belief state formula. As in CTL*, we use Φ, Ψ for state formula and ϕ, ψ for path formula. The formula ϵ is called belief state formula. In HMMs, we are uncertain about the current state, but we always know the current belief state. Therefore, we want to know if some (probabilistic) properties are valid in belief states. We consider the example in the introduction:

There is at least a 90 percent probability that the model produces a sequence of observations $O = (o_0, o_1, \dots, o_n)$.

This can be expressed by a belief state formula $\epsilon = \mathcal{P}_{\geq 0.9}(\mathbf{X}_{o_0} \mathbf{X}_{o_1} \dots \mathbf{X}_{o_n} tt)$. Intuitively, a belief state b satisfies ϵ if the probability measure w.r.t. b , i.e., Pr_b , of the set of paths satisfying $\mathbf{X}_{o_0} \mathbf{X}_{o_1} \dots \mathbf{X}_{o_n} tt$ meets the bound ≥ 0.9 . This formula corresponds the first objective function in Chapter 1 (Introduction). In *Speech Recognition* [25], we want to find out the most likely sentence given a language and some acoustic input. For example, if we know that the HMM for the word “Need” produces the acoustic observations with probability at least 0.9, we can almost conclude that this acoustic input represents the word “Need”. We indicate that this property cannot be expressed by any sublogics of POCTL* that we define in Section 4.3. Since such properties are very important for HMMs, we shall concentrate on the logic POCTL* and its model checking algorithm in Section 5.

For the sake of simplicity, we do not consider the exist operator. The formula $\exists \phi$ can “almost”¹ be replaced by the probability formula $\mathcal{P}_{>0} \phi$. The standard (i.e., unbounded) until formula is obtained by taking n equal to ∞ , i.e., $\phi \mathcal{U} \psi = \phi \mathcal{U}^{\leq \infty} \psi$. We use the abbreviations $\wedge, \diamond, \square$ which are defined in the same way as for CTL*. The timed variants of the temporal operators can be derived, e.g., $\diamond^{\leq n} \phi = tt \mathcal{U}^{\leq n} \phi$, $\square^{\leq n} \phi = \neg \diamond^{\leq n} \neg \phi$. In addition, we define the following abbreviation:

$$\bigvee_{i \in \{1, \dots, k\}} \phi_i \quad \text{for} \quad \phi_1 \vee \phi_2 \vee \dots \vee \phi_k$$

4.2 Semantics of POCTL*

Given the HMM \mathcal{H} , the semantics of POCTL* is defined by a satisfaction relation (denoted by \models) either between a state s and a state formula Φ , or between a path σ and a path formula ϕ , or between a belief state b and a belief state formula ϵ . We write $\mathcal{H}, s \models \Phi$, $\mathcal{H}, \sigma \models \phi$ and $\mathcal{H}, b \models \epsilon$ if state s , path σ and belief state b satisfy state formula Φ , path formula ϕ and belief state formula ϵ , respectively. If the model \mathcal{H} is clear from the context, we simply write $s \models \Phi$, $\sigma \models \phi$ and $b \models \epsilon$.

Definition 4.2.1. Semantics of POCTL*

Let $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ be an HMM with $s \in S$ and $\sigma \in \text{Path}$. Let b_s be the belief state with $b_s(s) = 1$ and $b_s(s') = 0$ for $s' \neq s$. The satisfaction relation \models is defined in Figure 4.1 where $\text{Pr}_b\{\sigma \in \text{Path} \mid \sigma \models \phi\}$, or $p_b(\phi)$ for short, denotes the probability measure of the set of all paths which satisfy ϕ and start with those state s with $b(s) > 0$. \square

A path satisfies the new operator $\mathbf{X}_o \phi$ if it starts with the observation o and the suffix² $\sigma[1]$ satisfies ϕ . The following lemma shows that the semantics of the probabilistic operator is well-defined.

Theorem 4.2.2. *Let ϕ be a POCTL* path formula, then, the set $\{\sigma \in \text{Path} \mid \sigma \models \phi\}$ is measurable in the probability space $(\text{Path}, \mathcal{F}, \text{Pr}_s)$ (or $(\text{Path}, \mathcal{F}, \text{Pr}_b)$) described in Lemma 3.4.6.*

Proof. See Appendix A. \square

Let Ω be a set of observations, i.e., $\Omega \subseteq \Theta$. We use the following useful abbreviations to shorten

¹As an example in which they do not coincide, we consider the formula $\square a$ in the HMM in Figure 3.3. The formula $\exists(\square a)$ is true because there exists one (and only one) path, i.e., the path contains only u (u^ω), which satisfies $\square a$. But the measure of the set containing only this path equals 0, hence, the formula $\mathcal{P}_{>0}(\square a)$ is not true.

²This suffix $\sigma[1]$ is well-defined for we have previously assumed that the model does not contain any absorbing states. If \mathcal{H} would contain an absorbing state s , there would be only one finite path starting with s , namely s itself. For this case, the next operator is not well-defined because the suffix $\sigma[1]$ does not exist.

$s \models a$	iff	$a \in L(s)$
$s \models \neg\Phi$	iff	$s \not\models \Phi$
$s \models \Phi \wedge \Psi$	iff	$s \models \Phi \wedge s \models \Psi$
$s \models \epsilon$	iff	$b_s \models \epsilon$
$\sigma \models \Phi$	iff	$\sigma_s[0] \models \Phi$
$\sigma \models \neg\phi$	iff	$\sigma \not\models \phi$
$\sigma \models \phi \wedge \psi$	iff	$\sigma \models \phi \wedge \sigma \models \psi$
$\sigma \models \mathbf{X}_o\phi$	iff	$\sigma_o[0] = o \wedge \sigma[1] \models \phi$
$\sigma \models \phi \mathcal{U}^{\leq n} \psi$	iff	$\exists 0 \leq j \leq n. (\sigma[j] \models \psi \wedge \forall i < j. \sigma[i] \models \phi)$
$b \models \mathcal{P}_{\leq p}(\phi)$	iff	$\text{Pr}_b\{\sigma \in \text{Path} \mid \sigma \models \phi\} \leq p$
$b \models \neg\epsilon$	iff	$b \not\models \epsilon$
$b \models \epsilon \wedge \epsilon'$	iff	$b \models \epsilon \wedge b \models \epsilon'$

Figure 4.1: Semantics of POCTL*

our notations:

$$\begin{aligned} \mathbf{X}_\Omega\phi & \quad \text{for} \quad \bigvee_{o \in \Omega} \mathbf{X}_o\phi \\ \phi_\Omega \mathcal{U}_\Omega^{\leq n} \psi & \quad \text{for} \quad (\phi \wedge \mathbf{X}_\Omega tt) \mathcal{U}^{\leq n-1} (\phi \wedge \mathbf{X}_\Omega \psi) \\ \phi_\Omega \mathcal{U}^{\leq n} \psi & \quad \text{for} \quad (\phi \wedge \mathbf{X}_\Omega tt) \mathcal{U}^{\leq n} \psi \end{aligned}$$

By the definition of $\mathbf{X}_\Omega\phi$, we obviously have $\sigma \models \mathbf{X}_\Omega\phi \equiv \sigma_o[0] \in \Omega \wedge \sigma[1] \models \phi$. The usual next operator can be described as $\mathbf{X}\phi \equiv \mathbf{X}_\Theta\phi$. Thus, the logic PCTL* can be considered as a sublogic of POCTL*.

For an HMM which contains absorbing states, we can transform it to an HMM which does not have absorbing states as described in Section 3.1 (Labeled Discrete-Time HMMs). Now, in parallel, we perform a transformation of the POCTL* formula ϕ by replacing each subformula $\mathbf{X}_o\psi$ by $\mathbf{X}_o(\psi \wedge \neg at_\dagger)$ and $\phi_1 \mathcal{U}^{\leq n} \phi_2$ by $\phi_1 \mathcal{U}^{\leq n} (\phi_2 \wedge \neg at_\dagger)$. Let ϕ' be the resulting POCTL* formula, it is easy to see that the interpretation of ϕ over the original model corresponds to the interpretation of ϕ' over the transformed model which does not contain absorbing states.

4.3 The Sublogics

Recall that the logic PCTL* is a combination of PCTL and QLS. In PCTL, arbitrary combinations of state formulas are possible, but the path formulas consists of only the next and until operators. The logic LTL allows arbitrary combinations of path formulas but only propositional state formulas. This section introduces the sublogics POCTL, OLTL and QOS of POCTL*. They can also be considered as extensions of the logics PCTL, LTL and QLS where the next operator is equipped with an observation (or a set of observations) constraint.

POCTL: We define the logic POCTL as a sublogic of POCTL* by imposing the restriction on POCTL* formulas such that every next and until operator ($\mathbf{X}, \mathcal{U}^{\leq n}$) should be immediately enclosed

in the probabilistic operator \mathcal{P} . Formally, POCTL is defined by:

$$\begin{aligned}\Phi &:= a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \epsilon \\ \phi &:= \mathbf{X}_\Omega \Phi \mid \Phi \mathcal{U}^{\leq n} \Phi \\ \epsilon &:= \mathcal{P}_{\triangleleft p}(\phi) \mid \neg\epsilon \mid \epsilon \wedge \epsilon\end{aligned}$$

where $\Omega \subseteq \Theta$ and $p \in [0, 1]$.

Since we have $\mathbf{X}\phi \equiv \mathbf{X}_\Theta\phi$, the logic PCTL is naturally a sublogic of POCTL. POCTL is a proper sublogic of POCTL*. For example, we let $a, a' \in AP$, then the formulas $\mathcal{P}_{<p}(\mathbf{X}\mathbf{X}a)$ and $\mathcal{P}_{<p}(a \mathcal{U}(\mathbf{X}a'))$ are not valid POCTL formulas, but are valid POCTL* formulas.

OLTL: In OLTL, we allow arbitrary combinations of path formulas, but only propositional state formulas. Formally, OLTL formulas are the path formulas defined by:

$$\phi := a \mid \neg\phi \mid \phi \wedge \phi \mid \mathbf{X}_o\phi \mid \phi \mathcal{U}^{\leq n}\phi$$

QOS: Recall that LTL can also be interpreted over probabilistic models [3] (Section 2.2). Now we extend it to QOS (quantitative OLTL specification) which shall contribute to POCTL* model checking.

A QOS formula is a pair $(\phi, \triangleleft p)$ where ϕ is an OLTL formula, $\triangleleft \in \{\leq, <, \geq, >\}$ and $p \in [0, 1]$. Let $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ be an HMM with $s \in S$. The semantics of the QOS formula is given by:

$$\mathcal{H}, s \models (\phi, \triangleleft p) \iff p_s(\phi) \triangleleft p$$

The logics OCTL* and OCTL can be defined as extensions of CTL* and CTL, in which the next operator is equipped with an observation, and a set of observations respectively. The semantics of the sublogics are intuitively clear from the interpretation of POCTL*.

Relationship of POCTL* and Its Sublogics: Figure 4.2 shows an overview of the relationship of the logic POCTL* and its sublogics. There is an arrow from a logic A to another logic B if A is a proper sublogic of B . The logics in the upper part can be considered as the probabilistic counterpart of the corresponding one in the lower part. Comparing to Figure 2.4, we conclude that the logic PCTL* (and the sublogics) is a sublogic of POCTL* (and the sublogics).

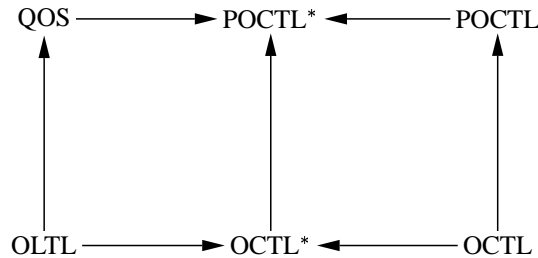


Figure 4.2: Relationship of the logic POCTL* and its sublogics

4.4 Specifying Properties in POCTL*

First, we indicate that we cannot calculate an exact probability by a POCTL* formula, however, we can specify a bound on the probability measure instead. Actually, we do not need the exact values in most cases. To illustrate the expressiveness of POCTL*, we consider some example properties for the coin toss model in Figure 3.1.

- *The probability that the next observation is head and then the model goes to state f (the fair coin) meets the bound < 0.2 .*

$$\mathcal{P}_{<0.2}(\mathbf{X}_{head}at_f)$$

This formula can be considered as a state formula or a belief state formula. A state (belief state) satisfies this formula if the probability is calculated using the measure w.r.t. the state (belief state).

- *The probability is at most 0.05, that we eventually get an observation head and then move to state f , whereas at any moment before we are either in state u_1 or state u_2 (the biased coins).*

$$\mathcal{P}_{\leq 0.05}((at_{u_1} \vee at_{u_2}) \cup \mathbf{X}_{head}at_f)$$

- *With probability at least 0.9, the model generates the observation sequence (o_0, o_1, \dots, o_n) .*

$$\mathcal{P}_{\geq 0.9}(\mathbf{X}_{o_0} \mathbf{X}_{o_1} \dots \mathbf{X}_{o_n} tt)$$

Note that we will see in Chapter 5 that the formula $\mathbf{X}_{o_0} \mathbf{X}_{o_1} \dots \mathbf{X}_{o_n} tt$ is called the characteristic formula of the cylinder set $\mathcal{C}(o_0, o_1, \dots, o_n)$.

- *The probability that the state sequence (s_0, s_1, \dots, s_n) produces the observation sequence (o_0, o_1, \dots, o_n) is at most 0.1.*

$$\mathcal{P}_{\leq 0.1}(s_0 \wedge \mathbf{X}_{o_0}(s_1 \wedge \mathbf{X}_{o_1}(\dots(s_n \wedge \mathbf{X}_{o_n} tt) \dots)))$$

For simplicity, we let s denote the atomic proposition at_s for a state s . The formula $s_0 \wedge \mathbf{X}_{o_0}(s_1 \wedge \mathbf{X}_{o_1}(\dots(s_n \wedge \mathbf{X}_{o_n} tt) \dots))$ is called the characteristic formula of the basic cylinder set $\mathcal{C}((s_0, o_0), \dots, (s_n, o_n))$ (see Chapter 5). See also the second objective function in Chapter 1 (Introduction).

- *The probability is at least 0.01, that we get an observation tail within 10 time units and at any moment before the state f produces the observation head.*

$$\mathcal{P}_{\geq 0.01}((at_f) \{_{head}\} \mathcal{U}_{\{_{tail}\}}^{\leq 10} tt)$$

Chapter 5

Model Checking

In this chapter, we present model checking algorithms for the logics POCTL*, POCTL and QOS. The model checking algorithm for POCTL* follows the same line as the one for PCTL* [23, 15, 14, 3] (Section 2.4). In more detail, first it will be reduced to the QOS model checking problem. The latter can further be reduced to a reach probability analysis. To that end, we construct a Büchi automaton for a given OLTL formula. This construction can be extended from the one presented by Gerth *et al.* [19] (Section 2.3). The POCTL model checking algorithm can be adapted from the one presented by Hansson & Jonsson [20].

5.1 POCTL* Formulas

Let $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ be an HMM with $s \in S$, and Φ be a POCTL* formula. The POCTL* model checking problem is to check whether $\mathcal{H}, s \models \Phi$ (or $s \models \Phi$ for short). The model checking algorithm for POCTL*, we shall present, is an adaption of the one presented in [23, 3].

The main idea of the POCTL* model checking algorithm is as follows. First, we identify the most deeply nested state subformula Ψ of Φ such that Ψ is not an atomic proposition. There are only three cases by the syntax of POCTL*, namely, Ψ is either a negation of an atomic proposition, or a conjunction of two atomic propositions, or a probabilistic operator. Second, we replace Ψ by a new atomic proposition $a_\Psi \notin AP$. Finally, we update the labeling function for all states satisfying Ψ , and replace all occurrences of Ψ in Φ by a_Ψ . The formula Φ will be recursively replaced by an equivalent state formula with smaller size, and finally is replaced by an atomic proposition a_Φ . Then, we only need to check whether $a_\Phi \in L(s)$.

The algorithm is shown in Figure 5.1. We let the line numbers in the following description refer to this algorithm. For the cases that Ψ is a negation of an atomic proposition (lines 3–7) or a conjunction of two atomic propositions (lines 8–12), we can easily determine whether $s \models \Psi$ for all states $s \in S$. The case that Ψ is the probabilistic operator $\mathcal{P}_{\leq p}(\phi)$ is more involved. We call the procedure SATQOS (line 14) with parameters \mathcal{H} (the model) and $(\phi, \leq p)$ (the QOS formula). Assuming that this procedure returns the set of state s with $s \models (\phi, \leq p)$, which is equivalent to $s \models \Psi$, we extend the label of such states by the new atomic proposition a_Ψ (lines 15–18). If Φ has not been replaced by an atomic proposition, it will be recursively processed (line 19). Otherwise, we check whether $a_\Phi \in L(s)$ (line 20).

Assuming that SATQOS works, the correctness of the algorithm is obvious. The procedure SATQOS will be discussed in Section 5.4 (QOS Formulas).

Example 5.1.1. Consider the HMM $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ depicted in Figure 5.2. The transition probability is given by $\mathbf{P}(s_4, s_4) = 1$, and $\mathbf{P}(s, s') = \frac{1}{2}$ otherwise. The labeling function L is

SATPOCTL*(\mathcal{H}, s, Φ)

- 1 Identify a most deeply nested state formula $\Psi \notin AP$.
- 2 **case** Ψ of
- 3 $\neg a$ where $a \in AP \implies$
- 4 $AP := AP \cup \{a_\Psi\}$; /* a_Ψ is a new atomic proposition */
- 5 **foreach** $s \in S$ such that $a \notin L(s)$ **do**
- 6 $L(s) := L(s) \cup \{a_\Psi\}$;
- 7 Replace all occurrences of Ψ in Φ by a_Ψ .
- 8 $a_1 \wedge a_2$ where $a_1, a_2 \in AP \implies$
- 9 $AP := AP \cup \{a_\Psi\}$; /* a_Ψ is a new atomic proposition */
- 10 **foreach** $s \in S$ such that $a_1 \in L(s) \wedge a_2 \in L(s)$ **do**
- 11 $L(s) := L(s) \cup \{a_\Psi\}$;
- 12 Replace all occurrences of Ψ in Φ by a_Ψ .
- 13 $\mathcal{P}_{\leq p}(\phi) \implies$
- 14 $Sat := \text{SATQOS}(\mathcal{H}, \phi, \leq p)$;
- 15 $AP := AP \cup \{a_\Psi\}$ /* a_Ψ is a new atomic proposition */
- 16 **foreach** $s \in Sat$ **do**
- 17 $L(s) := L(s) \cup \{a_\Psi\}$
- 18 Replace all occurrences of Ψ in Φ by a_Ψ .
- 19 **if** Φ has not been replaced by an atomic proposition **goto** 1.
- 20 **if** $a_\Phi \in L(s)$ **return** true **else return** false.

Figure 5.1: Model checking algorithm for POCTL*

indicated near the state (set braces are omitted). We let $\Theta = \{o_0, o_1, o_2\}$, and $\mu(s, o) = \frac{1}{3}$ for all states s and observations o . We apply the presented algorithm to check whether $s_0 \models \Phi$, where Φ equals $(\neg b) \wedge \mathcal{P}_{<0.05}(a \mathcal{U} \mathbf{X}_{o_0} b)$. First, we label states s_0, s_1, s_2, s_4 with the new atomic proposition $a_{\neg b}$. Then, we call the procedure $\text{SATQOS}(\mathcal{H}, a \mathcal{U} \mathbf{X}_{o_0} b, < 0.05)$ which returns the set $\{s_0, s_4\}$ satisfying the QOS formula (see Example 5.4.6). We update the label of the state s_0 by the new atomic proposition a_Ψ . Finally, we replace the conjunction of $a_{\neg b}$ and a_Ψ by the atomic proposition a_Φ and update the labeling functions of s_0 and s_4 by inserting a_Φ . As a result, we obtain $s_0 \models \Phi$. \square

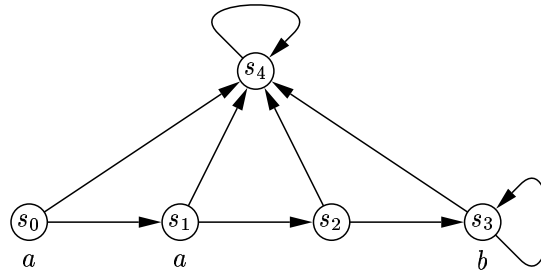


Figure 5.2: A simple hidden Markov model

Belief State: Now, we show how to check whether a belief state b satisfies a belief state formula ϵ , i.e., $b \models \epsilon$. The most interesting case is $\epsilon = \mathcal{P}_{\leq p}(\phi)$ where ϕ is a POCTL* path formula. By

definition,

$$b \models \mathcal{P}_{\leq p}(\phi) \iff p_b(\phi) \leq p \iff \sum_{s \in S} b(s) p_s(\phi) \leq p$$

therefore, it is sufficient to calculate $p_s(\phi)$ for all $s \in S$. Let Φ_1, \dots, Φ_k be the maximal state subformulas of ϕ . For $i = 1, \dots, k$, the set $Sat(\Phi_i)$ can be computed recursively using the POCTL* model checking algorithm in Figure 5.1. We replace the subformulas Φ_1, \dots, Φ_k by new atomic propositions a_1, \dots, a_k and update the labeling function by inserting a_i into $L(s)$ iff $s \in Sat(\Phi_i)$. The so obtained path formula ϕ' is an OLTL formula, and obviously we have $p_s(\phi) = p_s(\phi')$. In section 5.4 we will present how to calculate $p_s(\phi')$: It equals a reach probability in the product automaton of the HMM and the Büchi automaton for ϕ' .

5.2 POCTL Formulas

Let $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ be an HMM with $s \in S$, and Φ be a POCTL formula. The algorithm to check whether $s \models \Phi$ can be adapted from the one presented by Hansson & Jonsson [20] (Section 2.4). In case Φ is of the form $a, \neg\Phi', \Phi_1 \wedge \Phi_2, \mathcal{P}(\Phi_1 \mathcal{U}^{\leq n} \Phi_2), \mathcal{P}(\Phi_1 \mathcal{U} \Phi_2)$, the set $Sat(\Phi)$ can be determined using the same strategy as the one for PCTL. Let $p \in [0, 1]$, $\Omega \subseteq \Theta$ and $\leq \in \{\leq, <, \geq, >\}$. We only need to consider the case that $\phi = \mathcal{P}_{\leq p}(\mathbf{X}_\Omega \Phi')$. We observe that

$$p_s(\mathbf{X}_\Omega \Phi') = \mu_s(\Omega) \cdot \sum_{s' \in Sat(\Phi')} \mathbf{P}(s, s')$$

where $\mu_s(\Omega) = \sum_{o \in \Omega} \mu_s(o)$ and the set $Sat(\Phi') = \{s \in S \mid s \models \Phi'\}$ can be recursively evaluated. Thus, $s \models \mathcal{P}_{\leq p}(\mathbf{X}_\Omega \Phi')$ iff $p_s(\mathbf{X}_\Omega \Phi') \leq p$.

Example 5.2.1. Consider the coin toss model in Example 3.1.2. Suppose that we have seen coin tosses *head*, *tail*. Let Φ denote the formula $\mathcal{P}_{<0.2}(\mathbf{X}_{head}at_f)$ which means the probability that the next observation is *head* and then the model goes to state f meets the bound < 0.2 . Now we want to check whether $b_1 \models \Phi$. Recall that the belief state b_1 equals $(0.384, 0.216, 0.400)$ (see Example 3.2.3), thus,

$$\begin{aligned} p_{b_1}(\mathbf{X}_{head}at_f) &= \sum_{s \in S} b_1(s) \cdot p_s(\mathbf{X}_{head}at_f) \\ p_f(\mathbf{X}_{head}at_f) &= \mu_f(head) \cdot \mathbf{P}(f, f) = 0.40 \\ p_{u_1}(\mathbf{X}_{head}at_f) &= \mu_{u_1}(head) \cdot \mathbf{P}(u_1, f) = 0.08 \\ p_{u_2}(\mathbf{X}_{head}at_f) &= \mu_{u_2}(head) \cdot \mathbf{P}(u_2, f) = 0.04 \end{aligned}$$

We obtain $p_{b_1}(\mathbf{X}_{head}at_f) = 0.187$. Therefore, $b_1 \models \Phi$. □

5.3 Constructing Büchi Automata from OLTL Formulas

In this section, we explain how to construct a Büchi automaton that accepts exactly all infinite sequences satisfying a given OLTL formula ϕ . This shall be applied for QOS model checking in Section 5.4.

First, we assume that the given OLTL formula does not contain bounded until formulas. In Section 5.3.4, we will show how to deal with this restriction. Now we first give an interpretation of the OLTL formulas over the infinite words over the alphabet $\mathcal{P}(AP) \times \Theta$. For short, we let Σ denote the alphabet $\mathcal{P}(AP) \times \Theta$. Let $w = (w_0, o_0), (w_1, o_1), \dots \in \Sigma^\omega$ be an infinite word. We write

$w \models a$	iff	$a \in w_1[0]$ for $a \in AP$
$w \models \neg\phi$	iff	$w \not\models \phi$
$w \models \phi \wedge \psi$	iff	$w \models \phi \wedge w \models \psi$
$w \models \mathbf{X}_\Omega \phi$	iff	$w_2[0] \in \Omega \wedge w[1] \models \phi$
$w \models \phi \mathcal{U} \psi$	iff	$\exists j \geq 0. (w[j] \models \psi \wedge \forall 0 \leq i < j. w[i] \models \phi)$

Figure 5.3: Interpretation of OLTL over $\mathcal{P}(AP) \times \Theta$

$w_1[i] = w_i$, $w_2[i] = o_i$ and $w[i]$ for the suffix of w starting with (w_i, o_i) . The interpretation is given in Figure 5.3.

This construction is adapted from the one for LTL formulas introduced by Gerth *et al.* [19] (Section 2.3). The main idea is as follows. First, we transform the OLTL formula into normal form OLTL formula, i.e., we push all negations inside until they only precede atomic propositions. Afterwards, we construct a graph from the normal form OLTL formula. Then, we define a generalized Büchi automaton from the graph, and finally, the generalized Büchi automaton is transformed to a Büchi automaton. Now, we handle every step separately.

5.3.1 Normal Form OLTL Formulas

In Section 2.3, we dealt with the cases that an OLTL formula is of the form $\neg(\phi \vee \psi)$, $\neg(\phi \wedge \psi)$, $\neg(\phi \mathcal{U} \psi)$ and $\neg(\phi \mathcal{V} \psi)$. What we need additionally is to handle the operator $\mathbf{X}_\Omega \phi$ where $\Omega \subseteq \Theta$.

Lemma 5.3.1. *Let Ω be a subset of Θ , and ϕ be an OLTL formula. Then, $\neg \mathbf{X}_\Omega \phi \equiv \mathbf{X}_{\overline{\Omega}} \neg \phi \vee \mathbf{X}_\Omega \neg \phi$ where $\overline{\Omega}$ is the complement of Ω .*

Proof. Recall by definition of $\mathbf{X}_\Omega \phi$ we have $\sigma \models \mathbf{X}_\Omega \phi \equiv \sigma_o[0] \in \Omega \wedge \sigma[1] \models \phi$. Thus,

$$\begin{aligned} \sigma \models \neg \mathbf{X}_\Omega \phi &\iff \neg(\sigma \models \mathbf{X}_\Omega \phi) \iff \neg(\sigma_o[0] \in \Omega \wedge \sigma[1] \models \phi) \\ &\iff \sigma_o[0] \notin \Omega \vee \sigma[1] \not\models \phi \iff \sigma \models \mathbf{X}_{\overline{\Omega}} \neg \phi \vee \mathbf{X}_\Omega \neg \phi \end{aligned}$$

where $\mathbf{X} \neg \phi \equiv \mathbf{X}_\Omega \neg \phi \vee \mathbf{X}_{\overline{\Omega}} \neg \phi$, and obviously we have $\neg \mathbf{X}_\Omega \phi \equiv \mathbf{X}_{\overline{\Omega}} \neg \phi \vee \mathbf{X}_\Omega \neg \phi$. \square

We observe that the size of the resulting formula could be (in the worst case) $2|\phi|$.

5.3.2 Creating Graphs

The algorithm is depicted in Figure 5.4. We let the line numbers in the following description refer to this algorithm. A graph node is a tuple $Node = (Name, Father, Incoming, New, Old, Next, Observations)$. The fields *Father* and *Observations* are new fields (comparing to the one in Figure 2.6). For these new fields, we write $Father(Node) = Father$, and $Obser(Node) = Observations$. The nodes will be split during the construction. The field *Father* shall contain the name of the node from which the current one has been split (see lines 24-s30). This field is used for reasoning about the correctness of the algorithm only, and is not important for the construction. The other new field *Observations* is a subset of Θ . This set contains all possible observations to satisfy the formulas in *Old(Node)*. Assuming that at the end of the construction the formulas $\mathbf{X}_{\Omega_1} \phi_1, \dots, \mathbf{X}_{\Omega_k} \phi_k$ belong to *Old(Node)*. This implies that the formula $\mathbf{X}_{\Omega_1} \phi_1 \wedge \dots \wedge \mathbf{X}_{\Omega_k} \phi_k$ is valid in *Node*. Then, *Observations* is equal to the set $\bigcap_{i \in \{1, \dots, k\}} \Omega_i$ by observing that $\mathbf{X}_{\Omega_1} \phi_1 \wedge \mathbf{X}_{\Omega_2} \phi_2 \equiv$

```

EXPAND(Node, NodesSet)
1  if New(Node) =  $\emptyset$  then
2    if  $\exists N \in NodesSet$  with  $Old(N) = Old(Node)$  and  $Next(N) = Next(Node)$ 
3    then  $In(N) = In(N) \cup In(Node)$ ;
4      return (NodesSet);
5    else let name = NAME();
6      return (EXPAND((name, name, Name(Node), Next(Node),  $\emptyset$ ,  $\emptyset$ ,  $\Theta$ ),
7                    {Node}  $\cup$  NodesSet));
8  else
9    let  $\eta \in New(Node)$ ;
10    $New(Node) := New(Node) \setminus \{\eta\}$ ;
11   case  $\eta$  of
12      $\eta \in AP$  or  $\neg\eta \in AP$  or  $\eta = tt$  or  $\eta = ff \Rightarrow$ 
13       if  $\eta = ff$  or  $\neg\eta \in Old(Node)$  /* Current node contains a contradiction */
14       then return (NodesSet) /* Discard current node */
15       else  $Old(Node) := Old(Node) \cup \{\eta\}$ 
16         return (EXPAND(Node, NodesSet));
17      $\eta = \phi \wedge \psi \Rightarrow$ 
18       return (EXPAND((Name(Node), Father(Node), In(Node),
19                     New(Node)  $\cup$  ( $\{\phi, \psi\} \setminus Old(Node)$ ),
20                     Old(Node)  $\cup$   $\{\eta\}$ , Next(Node), Obser(Node), NodesSet));
21      $\eta = X_{\Omega}\phi \Rightarrow$ 
22       return (EXPAND((Name(Node), Father(Node), In(Node), New(Node),
23                     Old(Node)  $\cup$   $\{\eta\}$ , Next(Node)  $\cup$   $\{\phi\}$ , Obser(Node)  $\cap$   $\Omega$ , NodesSet));
24      $\eta = \phi \cup \psi$  or  $\phi \vee \psi$  or  $\phi \vee \psi \Rightarrow$ 
25       Node1 := (NAME(), Father(Node), In(Node),
26               New(Node)  $\cup$  ( $\{NEW1(\eta)\} \setminus Old(node)$ ),
27               Old(Node)  $\cup$   $\{\eta\}$ , Next(Node)  $\cup$   $\{NEXT1(\eta)\}$ , Obser(Node));
28       Node2 := (NAME(), Father(Node), In(Node),
29               New(Node)  $\cup$  ( $\{NEW2(\eta)\} \setminus Old(node)$ ),
30               Old(Node)  $\cup$   $\{\eta\}$ , Next(Node), Obser(Node));
31       return (EXPAND(Node2, EXPAND(Node1, NodesSet)));

CREATEGRAPH( $\phi$ )
32 let name = NAME();
33 return (EXPAND((name, name, {init},  $\{\phi\}$ ,  $\emptyset$ ,  $\emptyset$ ,  $\emptyset$ )));

```

Figure 5.4: The algorithm for constructing a graph for an OLTL formula

$X_{\Omega_1 \cap \Omega_2} \phi_1 \wedge \phi_2$. This corresponds to lines 21–23. The function NAME() generates a new string for each call, and the functions NEW1(η), NEW2(η) and NEXT1(η) are defined in Table 2.1.

For a normal form OLTL formula ϕ , the procedure CREATEGRAPH(ϕ) gives a set of nodes whose constructions are finished, i.e., the *New* field is empty. As in Section 2.3, we can build a graph $G = (V, E)$ where V is the nodes returned by the algorithm, and $(p, q) \in E$ if $p \in In(q)$.

Example 5.3.1. Let a and b be atomic propositions, and $\phi = a \cup Xb$. Applying the algorithm CREATEGRAPH(ϕ), we obtain the graph depicted in Figure 5.5. Below each node q_i , O , N and

$Obser$ indicate $Old(q_i)$, $Next(q_i)$ and $Obser(q_i)$, respectively. Comparing to Example 2.3.1, we additionally have the field $Obser$. \square

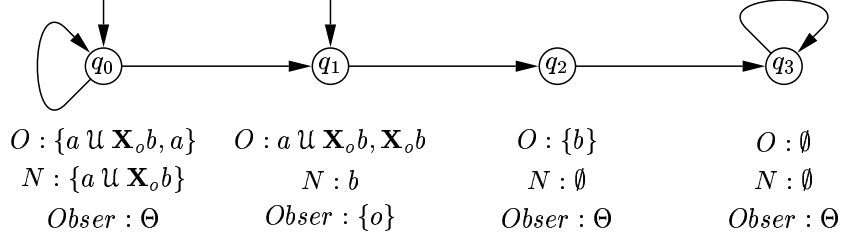


Figure 5.5: The graph for $a \cup \mathbf{X}_o b$

5.3.3 The Generalized Büchi Automaton

From the graph $G = (V, E)$ constructed in the last section, we define a generalized Büchi automaton $\mathcal{A}_\phi = (\Sigma, Q, L, \delta, Q_0, \mathcal{F})$ where Σ is the alphabet $\mathcal{P}(AP) \times \Theta$, and $L : Q \rightarrow \mathcal{P}(\mathcal{P}(AP)) \times \mathcal{P}(\Theta)$ is the labeling function. Other components are identical to the one described in Section 2.3.3.

Now we define the labeling function L . For a state q , we write $L(q) = (L_1(q), L_2(q))$ where $L_1(q)$ and $L_2(q)$ denote the first and second component of the label of q . Thus, $L_1(q)$ is a subset of $\mathcal{P}(A)$ and $L_2(q)$ is a subset of Θ . The first component $L_1(q)$ is defined as in Section 2.3, i.e., $L_1(q)$ contains all sets in $\mathcal{P}(AP)$ that are compatible with $Old(q)$. The second component $L_2(q)$ equals the set $Obser(q)$, i.e., the set of all possible observations to satisfy the formulas in $Old(q)$. The following theorem establishes the correspondence between OLTL formulas and generalized Büchi automata.

Theorem 5.3.2. The Generalized Büchi Automaton for the OLTL Formula ϕ

Let Σ denote the alphabet $\mathcal{P}(AP) \times \Theta$. The generalized Büchi automaton $\mathcal{A}_\phi = (\Sigma, Q, L, \delta, Q_0, \mathcal{F})$ constructed for the OLTL formula ϕ accepts exactly those infinite words over Σ that satisfy ϕ .

Proof. See Appendix B. \square

Using the same techniques as in Section 2.3, the generalized Büchi automaton can be transformed into an equivalent Büchi automaton. The following theorem shows the complexity of the construction:

Lemma 5.3.3. A Büchi automaton can be constructed for an OLTL formula ϕ with complexity $\mathcal{O}(k \cdot 2^{2|\phi|})$, where k is the number of the acceptance sets in the corresponding generalized Büchi automaton, and $k < |\phi|$.

Proof. First, we transform ϕ to a normal form OLTL formula. The size of the resulting formula could be maximal $2|\phi|$ because of the next operator. The theorem follows directly by applying Theorem 2.3.5. \square

Example 5.3.4. The Büchi automaton \mathcal{A}_ϕ for the OLTL formula $\phi = a \cup \mathbf{X}_o b$ is depicted in Figure 5.6. The states are represented by circles. $L_1(q)$ contains all subsets of AP which are compatible with the atomic propositions near the state q (comparing to Example 2.3.3). $L_2(q)$ contains all the observations near the state q ($L_2(q) = \Theta$ if there is no observation near the state q). For example, we have $L(q_0) = (\{\{a\}, \{a, b\}\}, \Theta)$, $L(q_1) = (\mathcal{P}(AP), \{o\})$ and $L(q_3) = (\mathcal{P}(AP), \Theta)$. The initial states can be identified by an incoming arrow, i.e., $Q_0 = \{q_0, q_1\}$. The final states are marked with a double circle, i.e., $\mathcal{F} = \{q_1, q_2, q_3\}$.

We consider the word $w = (\{a, b\}, *), (\{a\}, *), (\$, o), (\{b\}, *), (\$, *), (\$, *) \dots$ where $*$ represents arbitrary observations and $\$$ represents arbitrary subsets of atomic propositions. w is accepted by \mathcal{A}_ϕ since we have the corresponding execution $\pi = q_0, q_0, q_1, q_2, q_3, q_3, \dots$ such that $w_i \in L(\pi_i)$ and $\text{inf}(\phi) = \{q_3\}$ contains a final state. On the contrary, the word $w' = (\{a\}, *), (\{a\}, *), (\{a\}, *), (\{a\}, *), (\{a\}, *) \dots$ is not accepted. \square

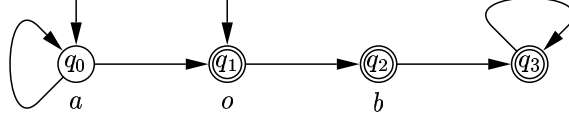


Figure 5.6: The generalized Büchi automaton for $a \mathcal{U} \mathbf{X}_o b$

5.3.4 The Bounded Until Formula

In this section, we show how to handle formulas involving the bounded until formula, i.e., formula of the form $\phi \mathcal{U}^{\leq n} \psi$ where $n \in \mathbb{N}$. The interpretation of this formula over the infinite word w is given by:

$$w \models \phi \mathcal{U}^{\leq n} \psi \quad \text{iff} \quad \exists 0 \leq j \leq n. (w[j] \models \psi \wedge \forall 0 \leq i < j. w[i] \models \phi)$$

By definition, we have $\phi \mathcal{U}^{\leq 0} \psi \iff \psi$, and for $n > 0$ we have $\phi \mathcal{U}^{\leq n} \psi \iff \psi \vee (\phi \wedge \mathbf{X}(\phi \mathcal{U}^{\leq n-1} \psi))$. To handle the negations of the bounded until formula, we first introduce the dual operator $\mathcal{V}^{\leq n}$ of $\mathcal{U}^{\leq n}$ by:

$$w \models \phi \mathcal{V}^{\leq n} \psi \quad \text{iff} \quad \forall 0 \leq j \leq n. (w[j] \models \psi \vee \exists 0 \leq i < j. w[i] \models \phi)$$

Then, we have $\phi \mathcal{V}^{\leq 0} \psi \iff \psi$, and for $n > 0$ we have $\phi \mathcal{V}^{\leq n} \psi \iff \psi \wedge (\phi \vee \mathbf{X}(\phi \mathcal{V}^{\leq n-1} \psi))$. By the semantics of $\mathcal{V}^{\leq n}$ of $\mathcal{U}^{\leq n}$, we use following rules to deal with the negations of the bounded until formula:

$$\neg(\phi \mathcal{U}^{\leq n} \psi) \longrightarrow (\neg\phi) \mathcal{V}^{\leq n} (\neg\psi) \quad \neg(\phi \mathcal{V}^{\leq n} \psi) \longrightarrow (\neg\phi) \mathcal{U}^{\leq n} (\neg\psi)$$

We consider the algorithm in Figure 5.4. If η is $\phi \mathcal{U}^{\leq n} \psi$ or $\phi \mathcal{V}^{\leq n} \psi$, we adapt the functions $\text{NEW1}(\eta)$, $\text{NEW2}(\eta)$ and $\text{NEXT1}(\eta)$ by:

η	$\text{NEW1}(\eta)$	$\text{NEXT1}(\eta)$	$\text{NEW2}(\eta)$
$\phi \mathcal{U}^{\leq n} \psi$	$\{\phi\}$	$\{\phi \mathcal{U}^{\leq n-1} \psi\}$	$\{\psi\}$
$\phi \mathcal{V}^{\leq n} \psi$	$\{\psi\}$	$\{\phi \mathcal{V}^{\leq n-1} \psi\}$	$\{\phi, \psi\}$
$\phi \vee \psi$	$\{\phi\}$	\emptyset	$\{\psi\}$

where $n > 0$. As an example, the bounded until formula $\phi \mathcal{U}^{\leq n} \psi$ in Figure 5.7 is split into two nodes.

5.4 QOS Formulas

This section presents the model checking algorithm for QOS formulas. The main structure of the algorithm is depicted in Figure 5.8. The input is an HMM $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ with $s \in S$ and a QOS formula $(\phi, \leq p)$ where $p \in [0, 1]$. We shall check whether $\mathcal{H}, s \models (\phi, \leq p)$. We first construct a Büchi automaton \mathcal{A}_ϕ for ϕ , then, we build the product automaton of the HMM \mathcal{H} and the Büchi automaton \mathcal{A}_ϕ . Finally, the problem to calculate the measure of paths in $\text{Path}^{\mathcal{H}}(s)$ satisfying ϕ is reduced to a reach probability analysis in the product automaton. The method we shall present is an adaption of the one introduced by Iyer & Narasimha [23] (Section 2.4).

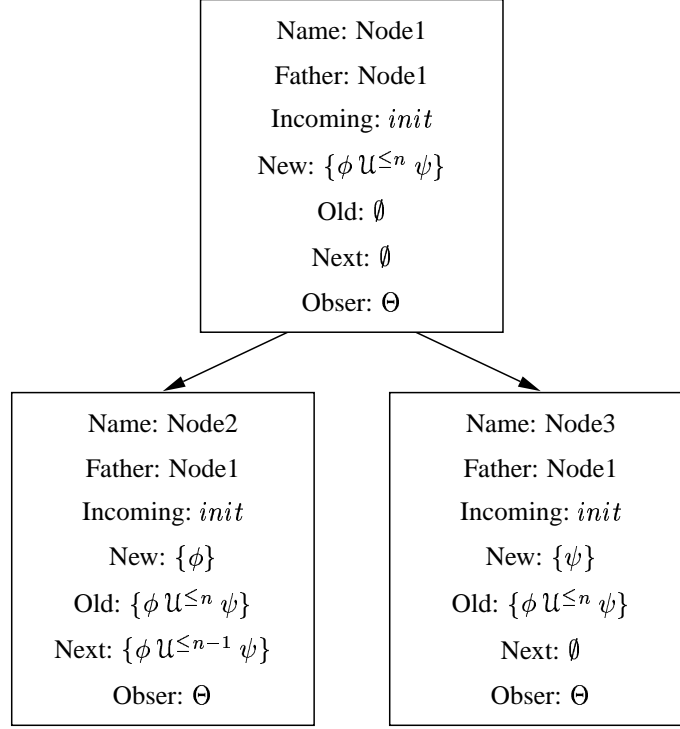


Figure 5.7: Splitting the bounded until formula

Definition 5.4.1. The Product of an HMM and a Büchi Automaton

Let $\mathcal{H} = (S, \mathbf{P}, L_{\mathcal{H}}, \Theta, \mu, \alpha)$ be an HMM, and $\mathcal{A}_{\phi} = (\Sigma, Q, L, \delta, Q_0, F)$ be the Büchi automaton for the OLTL formula ϕ . We define the product automaton $\mathcal{H} \times \mathcal{A}_{\phi} = (S_{\mathcal{A}}, \mathbf{P}_{\mathcal{A}}, L_{\mathcal{A}}, F_{\mathcal{A}}, \mu_{\mathcal{A}})$ by:

- $S_{\mathcal{A}} = \{(s, q) \in S \times Q \mid L_{\mathcal{H}}(s) \in L_1(q) \wedge \mu_s(L_2(q)) > 0\}$.
- $\mathbf{P}_{\mathcal{A}}((s, q), (s', q')) = \mathbf{P}(s, s') \cdot \mu_s(L_2(q))$ iff $q' \in \delta(q)$ and 0 otherwise.
- $L_{\mathcal{A}}((s, q)) = (L_{\mathcal{H}}(s), L_2(q))$.
- $F_{\mathcal{A}} = S \times F$.
- $\mu_{\mathcal{A}}((s, q), o) = \mu_s(o)$ if $o \in L_2(q)$ and 0 otherwise. □

We define a path $\sigma_{\mathcal{A}}$ over the product automaton as an infinite sequence $(s_0, o_0), (s_1, o_1), \dots \in (S_{\mathcal{A}} \times \Theta)^{\omega}$ with $\mu_{\mathcal{A}}(s_i, o_i) > 0$. It is possible that a state in the product automaton is absorbing. Since there is no (infinite) path which visits such a state, it can be omitted here.

Recall that a BSCC of a graph is an SCC in which every state that is reachable from a state of the SCC is in the SCC. We define the projection of a state $(s, q) \in S_{\mathcal{A}}$ onto \mathcal{H} as s . Similarly, we define the projection of paths and SCCs.

Lemma 5.4.2. *Let $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ be an HMM, and let Z denote the set of paths which do not loop in a BSCC or which visit a strictly subset of a BSCC in \mathcal{H} . Then, the probability measure (w.r.t. a state or a belief state) of Z is 0.*

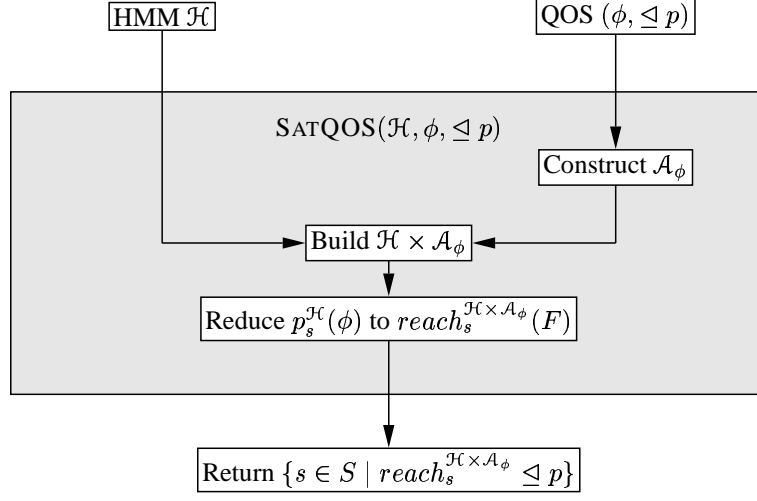


Figure 5.8: QOS Model checking

Proof. Let $\mathcal{D} = (S, \mathbf{P}, L)$ be the underlying DTMC of \mathcal{H} . Let $Z_{\mathcal{D}}$ denote the set of paths which do not loop in a BSCC or which visit strictly a subset of a BSCC in \mathcal{D} . By the fundamental property of Markov chains [27, 15], the probability measure of $Z_{\mathcal{D}}$ is 0. It is obvious that Z induces $Z_{\mathcal{D}}$ (where we omit the observations) and the measures of them are equal. \square

The previous lemma suggests that we only need to concentrate on those paths which visit all of the states of a BSCC in \mathcal{H} infinitely often. Now we reduce the QOS model checking problem to a reach probability analysis in the product automaton. In what follows, we let $\mathcal{H} \times \mathcal{A}_\phi$ denote the product automaton of the HMM \mathcal{H} and the Büchi automaton \mathcal{A}_ϕ where ϕ is an OLTTL formula.

Lemma 5.4.3. *Let $\sigma \in Path^{\mathcal{J}^c}$ with $\sigma \models \phi$, and let σ loop in a BSCC B . Then, there exists a path $\sigma_{\mathcal{A}}$ which loops in an SCC $B_{\mathcal{A}}$ in $\mathcal{H} \times \mathcal{A}_\phi$ such that*

- $B_{\mathcal{A}}$ contains at least one final state (s, q_f) , such that $\sigma_{\mathcal{A}}$ visits (s, q_f) infinitely often.
- $\sigma_{\mathcal{A}}$ projects to σ onto $Path^{\mathcal{J}^c}$.
- $B_{\mathcal{A}}$ projects to B onto \mathcal{H} .

Proof. Let $\sigma = (s_0, o_0), (s_1, o_1), \dots$. Let w be the word $(L(s_0), o_0), (L(s_1), o_1), \dots$. Thus, w satisfies ϕ . By Theorem 5.3.2, there exists an execution $\pi = q_0, q_1, q_2, \dots$ in the Büchi automaton \mathcal{A}_ϕ such that $(L(s_i), o_i) \in L(q_i)$ for all $i \in \mathbb{N}$ and $\text{inf}(\pi)$ contains at least one final state $q_f \in F$. We let $\sigma_{\mathcal{A}} = ((s_0, q_0), o_0), ((s_1, q_1), o_1), \dots$, then $\sigma_{\mathcal{A}}$ is a path in $\mathcal{H} \times \mathcal{A}_\phi$. Define $B_{\mathcal{A}} = \text{inf}(\sigma_{\mathcal{A}})$, therefore, $B_{\mathcal{A}}$ contains at least (since S is finite) one final state. Obviously, $B_{\mathcal{A}}$ is an SCC which projects to the BSCC B onto \mathcal{H} . \square

Lemma 5.4.4. *Let $\sigma_{\mathcal{A}} = ((s_0, q_0), o_0), ((s_1, q_1), o_1), \dots$ be a path in $\mathcal{H} \times \mathcal{A}_\phi$ such that $q_0 \in Q_0$, and let $\sigma_{\mathcal{A}}$ loop in an SCC $B_{\mathcal{A}}$ in $\mathcal{H} \times \mathcal{A}_\phi$. Furthermore, let $B_{\mathcal{A}}$ contain at least one final state (s, q) , and let $B_{\mathcal{A}}$ project to a BSCC B in \mathcal{H} . Then, the path $\sigma = (s_0, o_0), (s_1, o_1), \dots$ loops in B and satisfies ϕ .*

Proof. Obviously, the induced infinite sequence $\pi = q_0, q_1, \dots$ in \mathcal{A}_ϕ is an accepting execution. Now we define a word $w = (L(s_0), o_0), (L(s_1), o_1), \dots$. Since $\sigma_{\mathcal{A}}$ is a path over the product automaton,

$(L(s_i), o_i) \in L(q_i)$. This implies that w is accepted by \mathcal{A}_ϕ . By Theorem 5.3.2 we obtain $w \models \phi$, thus, $\sigma \models \phi$ where σ is the path $(s_0, o_0), (s_1, o_1) \dots$ in \mathcal{H} . Obviously, σ loops in the BSCC B in \mathcal{H} . \square

Recall we let $p_s(\phi)$ denote $\Pr_s\{\sigma \in \text{Path}^{\mathcal{H}}(s) \mid \sigma \models \phi\}$. Now we get the following theorem:

Theorem 5.4.5. *The measure $p_s(\phi)$ is equal to the measure of the set of the projections onto \mathcal{H} of paths $\sigma_{\mathcal{A}}$ such that $\sigma_{\mathcal{A}}$ loops in an SCC containing at least one final state, and $\sigma_{\mathcal{A}}$ starts with state (s, q_0) where $q_0 \in Q_0$ is an arbitrary initial state in \mathcal{A}_ϕ . \square*

Based on the preceding theorem, Figure 5.9 gives an algorithm for computing the set of states which satisfy the QOS formula $(\phi, \trianglelefteq p)$. In the second step¹, it is possible that an SCC B_1 and a BSCC B_2 in $\mathcal{H} \times \mathcal{A}_\phi$ project to the same BSCC onto \mathcal{H} . In this case we only choose the BSCC B_2 (e.g., see Appendix C). In step 4, we should take care that no two paths have the same projection. Since the constructed Büchi automaton is nondeterministic, it is possible that two paths have the same projection. In this case, we take only one of them (e.g., see Appendix C).

SATQOS($\mathcal{H}, \phi, \trianglelefteq p$)

- 1 Construct the Büchi automaton \mathcal{A}_ϕ and product automaton $\mathcal{H} \times \mathcal{A}_\phi$.
- 2 Compute the set \mathcal{B} of SCCs of the graph $\mathcal{H} \times \mathcal{A}_\phi$ such that every $B_{\mathcal{A}} \in \mathcal{B}$ contains at least one final state and projects to a BSCC onto \mathcal{H} .
- 3 Replace each SCC $B_{\mathcal{A}} \in \mathcal{B}$ by a new node b_i to get a new graph.
- 4 Let $p_{(s, q_0)}$ denote the probability to reach \mathcal{B} from state (s, q_0) . For every $s \in S$ we calculate the probability $p_s = \sum_{q_0 \in Q_0} p_{(s, q_0)}$ taking care that no two paths have the same projection in \mathcal{H} .
- 5 Return $\{s \in S \mid p_s \trianglelefteq p\}$.

Figure 5.9: Model checking algorithm for QOS

Example 5.4.6. The product automaton of the HMM in Example 5.1.1 (Figure 5.2) and the Büchi automaton² in Example 5.3.4 (Figure 5.6) is depicted in Figure 5.10. This is only a part of the product automaton which is sufficient to calculate the probability measure w.r.t. state s_0 . This part automaton is constructed by need (see Section 5.5 or Figure 5.11). States are omitted if they are either absorbing or not reachable from states (s_0, q_0) and (s_0, q_1) . We observe that there is only one BSCC $B = \{(s_4, q_3)\}$ which contains a final state and projects to a BSCC onto the HMM. The probability to reach B from state (s_3, q_3) is 1 and thus also 1 from state (s_3, q_2) . Then, the probability to reach B from state (s_0, q_0) is:

$$\begin{aligned} & \mathbf{P}_{\mathcal{A}}((s_0, q_0), (s_1, q_1)) \cdot \mathbf{P}_{\mathcal{A}}((s_1, q_1), (s_2, q_1)) \cdot \mathbf{P}_{\mathcal{A}}((s_2, q_1), (s_3, q_2)) \\ &= \mathbf{P}(s_0, s_1) \cdot \mathbf{P}(s_1, s_2) \cdot \mu_{s_2}(o) \cdot \mathbf{P}(s_2, s_3) = \frac{1}{24} = 0.042 \end{aligned}$$

Therefore, for $\Phi = \mathcal{P}_{<0.05}(a \mathcal{U} \mathbf{X}_{o_0} b)$ we have that $s_0 \models \Phi$. In a similar way we get that $s_4 \models \Phi$, and $s \not\models \Phi$ if $s \in \{s_1, s_2, s_3\}$. \square

¹Recall that in [23] Iyer & Narasimha considered the BSCCs instead of SCCs in the product automaton. But to prove the correctness, they used a lemma which says that if a BSCC B in the DTMC is the projection of an SCC B' in the product automaton, B' is a BSCC. This is wrong. Appendix C contains a counterexample.

²Actually, we have constructed a generalized Büchi automaton $\mathcal{A}_\phi = (\Sigma, Q, L, \delta, Q_0, \mathcal{F})$ in Example 5.1.1, where the acceptance set \mathcal{F} equals $\{q_1, q_2, q_3\}$. Since there is only one set in \mathcal{F} , we obtain the corresponding Büchi automaton $\mathcal{A}_\phi = (\Sigma, Q, L, \delta, Q_0, F)$ where $F = \{q_1, q_2, q_3\}$.

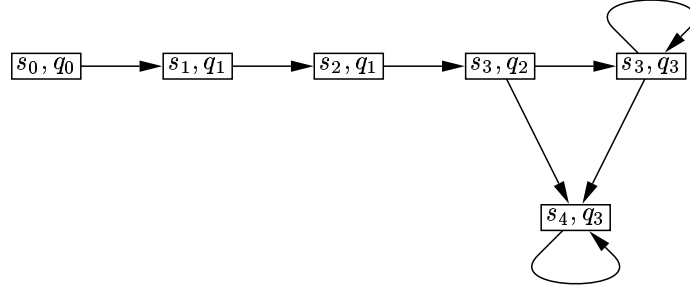


Figure 5.10: The product automaton

5.5 Improving the Efficiency

In this section, we give efficient algorithms for some special POCTL* formulas. After that we give some further improvements.

The Formula $s_0 \wedge \mathbf{X}_{o_0}(s_1 \wedge \mathbf{X}_{o_1}(\dots(s_n \wedge \mathbf{X}_{o_n} tt) \dots))$: For state $s \in S$, we let at_s denote the atomic propositions which asserts that the model resides in state s . If it is clear from the context, we simply abbreviate at_s by s , i.e., the state itself. Given a basic cylinder set $\mathcal{C}((s_0, o_0), \dots, (s_n, o_n))$, we define a formula $\phi = s_0 \wedge \mathbf{X}_{o_0}(s_1 \wedge \mathbf{X}_{o_1}(\dots(s_n \wedge \mathbf{X}_{o_n} tt) \dots))$ which is called the characteristic formula of this basic cylinder set. The following lemma shows the relation between this formula and the corresponding cylinder set.

Lemma 5.5.1. The Characteristic Formula of $\mathcal{C}((s_0, o_0), \dots, (s_n, o_n))$

Let $\phi = s_0 \wedge \mathbf{X}_{o_0}(s_1 \wedge \mathbf{X}_{o_1}(\dots(s_n \wedge \mathbf{X}_{o_n} tt) \dots))$ where $s_0, \dots, s_n \in S$ and $o_0, \dots, o_n \in \Theta$. Then, $\{\sigma \in Path \mid \sigma \models \phi\} = \mathcal{C}((s_0, o_0), \dots, (s_n, o_n))$. We call ϕ the characteristic formula of the basic cylinder set $\mathcal{C}((s_0, o_0), \dots, (s_n, o_n))$.

Proof. By induction on n .

Basis of induction $n=0$:

$$\begin{aligned} \sigma \models \phi &\iff \sigma \models s_0 \wedge \mathbf{X}_{o_0} tt \iff \sigma_s[0] = s_0 \wedge \sigma \models \mathbf{X}_{o_0} tt \\ &\iff \sigma_s[0] = s_0 \wedge \sigma_o[0] = o_0 \iff \sigma \in \mathcal{C}((s_0, o_0)) \end{aligned}$$

Induction step $n \Rightarrow n+1$:

$$\begin{aligned} \sigma \models \phi &\iff \sigma \models s_0 \wedge \mathbf{X}_{o_0}(s_1 \wedge \mathbf{X}_{o_1}(\dots(s_{n+1} \wedge \mathbf{X}_{o_{n+1}} tt) \dots)) \\ &\iff \sigma_s[0] = s_0 \wedge \sigma_o[0] = o_0 \wedge \sigma[1] \models s_1 \wedge \mathbf{X}_{o_1}(\dots(s_{n+1} \wedge \mathbf{X}_{o_{n+1}} tt) \dots) \\ &\stackrel{ih}{\iff} \sigma_s[0] = s_0 \wedge \sigma_o[0] = o_0 \wedge \sigma[1] \in \mathcal{C}((s_1, o_1), \dots, (s_{n+1}, o_{n+1})) \\ &\iff \sigma \in \mathcal{C}((s_0, o_0), \dots, (s_{n+1}, o_{n+1})) \end{aligned}$$

where *ih* denotes induction hypothesis. □

Let $s \in S$ and $C = \mathcal{C}((s_0, o_0), \dots, (s_n, o_n))$. By Equation 3.8 we know that the probability measure (w.r.t. Pr_s) of the basic cylinder set C equals $\mathbf{i}(s, s_0) \mu_{s_0}(o_0) \prod_{i=1}^n \mathbf{P}(s_{i-1}, s_i) \mu_{s_i}(o_i)$, where $\mathbf{i}(s, s_0) = 1$ if $s = s_0$, and $\mathbf{i}(s, s_0) = 0$ if $s \neq s_0$. By the preceding lemma, to check whether $s \models \mathcal{P}_{\leq p}(\phi)$ boils down to checking whether the probability measure of the basic cylinder set, i.e., $\text{Pr}_s(C)$, meets the bound $\leq p$.

Example 5.5.2. We consider the coin toss model in Figure 3.1. Let $\phi = f \wedge \mathbf{X}_{head}(u_1 \wedge \mathbf{X}_{tail}tt)$ be the characteristic formula of the basic cylinder set $\mathcal{C}((f, head), (u_1, tail))$. Obviously,

$$\begin{aligned} \Pr_f(\mathcal{C}((f, head), (u_1, tail))) &= \mu_f(head) \cdot \mathbf{P}(f, u_1) \cdot \mu_{u_1}(tail) \\ &= 0.5 * 0.1 * 0.2 = 0.01 \end{aligned}$$

As a result, $s \not\models \mathcal{P}_{>0.05}(\phi)$.

The Formula $\mathbf{X}_{o_0}\mathbf{X}_{o_1}\dots\mathbf{X}_{o_n}tt$: We define a path formula $\phi = \mathbf{X}_{o_0}\mathbf{X}_{o_1}\dots\mathbf{X}_{o_n}tt$ given the cylinder set $\mathcal{C}(o_0, \dots, o_n)$. The following lemma is obvious:

Lemma 5.5.3. The Characteristic Formula of $\mathcal{C}(o_0, \dots, o_n)$

Let $\phi = \mathbf{X}_{o_0}\mathbf{X}_{o_1}\dots\mathbf{X}_{o_n}tt$ where $o_0, \dots, o_n \in \Theta$. Then, $\{\sigma \in Path \mid \sigma \models \phi\} = \mathcal{C}(o_0, \dots, o_n)$. We call ϕ the characteristic formula of the cylinder set $\mathcal{C}(o_0, \dots, o_n)$.

Proof. Similar to Lemma 5.5.1. □

Let α be the initial distribution. The preceding lemma and Lemma 3.4.5 imply that to check whether $\alpha \models \mathcal{P}_{\leq p}(\phi)$ boils down to checking $\sum_{s \in S} \alpha(s) \Pr_s(\mathcal{C}(o_0, \dots, o_n)) = \sum_{s \in S} \alpha_n(s) \leq p$. Applying Equation 3.3 and 3.4, the value $\alpha_n(s)$ can be solved inductively with complexity $\mathcal{O}(|S|^{2n})$ (see [33]).

Example 5.5.4. We consider the coin toss model in Figure 3.1. Let $\phi = \mathbf{X}_{head}\mathbf{X}_{tail}tt$ be the characteristic formula of the cylinder set $\mathcal{C}(head, tail)$. The initial distribution α is $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. By Example 3.2.3 we have $\alpha_0 = \alpha D_{head} = (\frac{1}{6}, \frac{4}{15}, \frac{2}{15})$. Applying Equation 3.5:

$$\alpha_0 \mathbf{P}D_{tail} = \left(\frac{1}{6}, \frac{4}{15}, \frac{2}{15} \right) \begin{pmatrix} 0.8 & 0.1 & 0.1 \\ 0.1 & 0.8 & 0.1 \\ 0.1 & 0.1 & 0.8 \end{pmatrix} \begin{pmatrix} 0.5 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.6 \end{pmatrix} = (0.087, 0.049, 0.090)$$

Therefore, $\alpha_1 = (0.087, 0.049, 0.090)$. Summing over all states we have $\sum_{s \in S} \alpha_1(s) = 0.226 > 0.2$, and thus, $\alpha \models \mathcal{P}_{\geq 0.2}(\phi)$.

Building the Automaton by Need: By Definition 5.4.1, the set of states of the product automaton contains all pairs $(s, q) \in S \times Q$. In case Φ is a simple probabilistic operator, i.e., $\mathcal{P}_{\leq p}(\phi)$ where there is no probabilistic operator in ϕ , we give a more efficient algorithm to check whether $\mathcal{H}, s \models \Phi$. This is equivalent to check whether $\mathcal{H}, s \models (\phi, \leq p)$. We observe that many states of the product automaton might not be reachable from initial states (s, q_0) where $q_0 \in Q_0$. To avoid this, we construct the states of the product automaton as needed. Let $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ be an HMM with $s_0 \in S$, $\mathcal{A}_\phi = (\Sigma, Q, L, \delta, Q_0, F)$ be a Büchi automaton for an OLTTL formula ϕ . To simplify the notations, we shall represent the transition function \mathbf{P}_A as a set of tuples $((s, q), p, (s', q'))$ where $s, s' \in S, q, q' \in Q$ and $p \in (0, 1]$.

The algorithm starts with the unprocessed set unp which is initially the set of initial states (lines 1–2). For every state in unp , all reachable successors (lines 4–9) are obtained, and the transition is added to the transition matrix (line 10). The successor is inserted into unp if it has not been already processed (lines 11–12). As an example, we consider the product automaton depicted in Figure 5.2 which is constructed by need. Note that the states, which are either absorbing, or not reachable from an initial state, are omitted.

```

BUILDPRODUCT( $\mathcal{H}, \mathcal{A}_\phi, s_0$ )
1   $S_{\mathcal{A}} := \emptyset; \mathbf{P}_{\mathcal{A}} := \emptyset$ 
2   $unp := \{(s_0, q) \mid q \in Q_0 \wedge L(s_0) \in L_1(q) \wedge \mu_{s_0}(L_2(q)) > 0\}$ 
3  while  $unp \neq \emptyset$  do
4    let  $(s, q) \in unp$ 
5     $unp := unp \setminus \{(s, q)\}$ 
6     $S_{\mathcal{A}} := S_{\mathcal{A}} \cup \{(s, q)\}$ 
7    for each  $s' \in S$  with  $\mathbf{P}(s, s') > 0$  do
8      for each  $q' \in \delta(q)$  do
9        if  $L(s') \in L_1(q')$ 
10          $\mathbf{P}_{\mathcal{A}} := \mathbf{P}_{\mathcal{A}} \cup \{((s, q), \mathbf{P}(s, s') \cdot \mu_s(L_2(q)), (s', q'))\}$ 
11         if  $(s', q') \notin unp$ 
12            $unp := unp \cup (s', q')$ 

```

Figure 5.11: Build the reachable part of the product automaton

Rewriting the Formula: To construct the Büchi automaton for an OLTL formula ϕ , we first transform ϕ to a normal form OLTL formula. In this step, we can rewrite the formula using some equivalence transformation. For example, if $\Omega = \emptyset$, $\mathbf{X}_\Omega \phi$ equals ff , and if $\Omega = \Theta$, $\mathbf{X}_\Omega \phi$ equals $\mathbf{X}\phi$. Another example:

$$\neg \mathbf{X}_{\Omega_1} \phi \wedge \mathbf{X}_{\Omega_2} \psi = (\mathbf{X}_{\overline{\Omega_1}} tt \vee \mathbf{X}_{\Omega_1} \neg \phi) \wedge \mathbf{X}_{\Omega_2} \psi = \mathbf{X}_{\overline{\Omega_1} \cap \Omega_2} \psi \vee \mathbf{X}_{\Omega_1 \cap \Omega_2} \neg \phi \wedge \psi$$

where $\Omega_1, \Omega_2 \subseteq \Theta$.

Moving the Negations to the Bound: Let ϕ be an OCTL formula, then the formula $\mathcal{P}_{\triangleleft p}(\neg \phi)$ is equivalent to the formula $\mathcal{P}_{\overline{\triangleleft} p}(\phi)$ where $\overline{\leq} = \geq$, $\overline{\geq} = \leq$, $\overline{\prec} = \succ$ and $\overline{\succ} = \prec$. Thus, it is equivalent to check $s \models \mathcal{P}_{\triangleleft p}(\neg \phi)$ and to check $s \models \mathcal{P}_{\overline{\triangleleft} p}(\phi)$. In the case that ϕ is a nested operator, such as $\mathbf{X}_{o_0} \mathbf{X}_{o_1} \dots \mathbf{X}_{o_n} tt$, we do not need to push the negations which would generate a formula of size $2n$.

Reducing to POCTL Model Checking: Since the POCTL model checking algorithm is more efficient, we can use it to deal with QOS formulas of the form $(\phi \mathcal{U} \psi, \triangleleft p)$ (or $(\phi \mathcal{U}^{\leq n} \psi, \triangleleft p)$) where ϕ and ψ are POCTL* path formulas which can be verified recursively.

5.6 Complexity of the Model Checking Algorithms

In this section, we discuss the complexity of the POCTL* model checking algorithm presented in Section 5.1 (see Figure 5.1). First, we analyse the complexity of the procedure SATQOS in Section 5.4 (see Figure 5.8). We let k denote the number of the acceptance sets as described in Lemma 5.3.3.

Lemma 5.6.1. [23] *The complexity of each call of SATQOS($\mathcal{H}, \phi, \triangleleft p$) is $\mathcal{O}(|\mathcal{H}| \cdot k \cdot 2^{2|\phi|})^3$.*

Proof. By Lemma 5.3.3, the Büchi automaton \mathcal{A}_ϕ has (in the worst case) $\mathcal{O}(k \cdot 2^{2|\phi|})$ states. Thus, the product automaton $\mathcal{H} \times \mathcal{A}_\phi$ has states $\mathcal{O}(|\mathcal{H}| \cdot k \cdot 2^{2|\phi|})$. The complexity of the algorithm SATQOS is dominated by step 4 where a depth-first search needs to be carried out for each node of $\mathcal{H} \times \mathcal{A}_\phi$. The complexity of the depth-first search [23] is cubic in the size of the product automaton and this concludes the proof. \square

Theorem 5.6.2. *The complexity of the algorithm SATPOCTL* is $\mathcal{O}(|\phi| \cdot (|\mathcal{H}| \cdot k \cdot 2^{2|\phi|})^3)$.*

Proof. Since there are at most $|\phi|$ iterations in the algorithm SATPOCTL*, and at every iteration the worst case complexity is $\mathcal{O}((|\mathcal{H}| \cdot k \cdot 2^{2|\phi|})^3)$ by the preceding lemma. \square

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we have defined probability spaces (w.r.t. state and belief state) for a given HMM. Then we have presented the temporal logic POCTL* with which we can specify state-based, path-based and belief state-based properties over HMMs. With POCTL* one can specify properties not only over the underlying DTMC, but also over the set of processes producing observations. The latter are very important in *Speech Recognition* [25] in which we want to find out the most likely sentence (with the highest score) given a language and some acoustic input (observations). As we have shown that properties, such as

The HMM \mathcal{H} (e.g., for the word “Need”) produces the (acoustic) observations $O = (o_0, \dots, o_n)$ with probability at least 0.9,

can be expressed by POCTL*, but not by its sublogics.

Finally, we have focused on the POCTL* model checking algorithm. The most interesting case is to deal with the probabilistic operator, and we have shown that this can be reduced to QOS model checking. To that end, we have presented how to construct a Büchi automaton from an OLTTL formula. Then, the QOS model checking problem is reduced to a reach probability analysis in the product automaton of the HMM and the constructed Büchi automaton. The complexity of our model checking algorithm is polynomial in the size of the model and exponential in the length of the formula.

6.2 Future Work

In this section, we consider some interesting directions for future work.

HMDP: An HMM can be extended to a Hidden Markov decision process (HMDP) [10, 17] such that probabilistic and nondeterministic choices coexist. In an HMM, a successor of a state s is selected probabilistically according to the transition matrix. On the contrary, in an HMDP, for a state s , we first select a probabilistic distribution over actions nondeterministically. Then, a successor of s can be chosen probabilistically according to the selected distribution over actions.

The nondeterminism is resolved by *schedulers* [6] (called *strategy* in [10, 17], *adversary* in [3] and *policy* in [32]). A scheduler η assigns a distribution over actions to a finite sequence of states (history). Given a scheduler η , one can select a successor of a state probabilistically, as in an HMM.

Moreover, we can get a probability measure [10] \Pr_s^η w.r.t. the scheduler η and a state s . Thus, the logic POCTL* can be extended to interpret properties over HMDPs in the following way:

$$s \models \mathcal{P}_{\leq p}(\phi) \quad \text{iff} \quad \forall \eta. \Pr_s^\eta \{ \sigma \in Path^\eta \mid \sigma \models \phi \} \leq p$$

Since a belief state is a distribution over states, we can extend the probability measure w.r.t. s and η to the one w.r.t. a belief state and η . The semantics that a belief state satisfies a belief state formula can also be extended in a similar way.

De Alfaro presented a PCTL* model checking algorithm over MDPs. The main idea is similar to the POCTL* model checking algorithm over HMMs. Namely, one constructs the product automaton of an MDP and a deterministic Büchi automaton [3] which can be obtained from the nondeterministic Büchi automaton. The key step is also to reduce the probability measure in the MDP to a reach probability analysis in the product automaton which can be recursively expressed by the following equation:

$$reach_s(U) = \max_{a \in Act(s)} \left(\sum_{s' \notin U} \mathbf{P}(s, a, s') reach_{s'}(U) + \sum_{s' \in U} \mathbf{P}(s, a, s') \right)$$

where U is a subset of states in the product automaton. One could extend this for a POCTL* model checking algorithm over HMDPs as follows:

$$reach_s(U) = \max_{a \in Act(s)} \left(\sum_{s' \notin U} \mathbf{P}_{\mathcal{A}}(s, a, s') reach_{s'}(U) + \sum_{s' \in U} \mathbf{P}_{\mathcal{A}}(s, a, s') \right)$$

HMDP with Fairness: Baier [3] extended the logic PCTL* to interpret properties over concurrent probabilistic systems (similar to MDPs) with fairness assumptions. She also presented a PCTL* model checking algorithm over concurrent probabilistic systems with fairness assumptions which is adapted from the one by De Alfaro. It could be extended to a POCTL* model checking algorithm over HMDPs with fairness assumptions.

Appendix A

Proof of Theorem 4.2.2

Theorem 4.2.2. *Let ϕ be a POCTL* path formula, then, the set $\{\sigma \in Path \mid \sigma \models \phi\}$ is measurable in the probability space $(Path, \mathcal{F}, Pr_s)$ (or $(Path, \mathcal{F}, Pr_b)$) described in Lemma 3.4.6.*

We first show a lemma to prepare for the proof. For $n \in \mathbb{N} \cup \{\infty\}$, we let I_n denote the set $\{0, \dots, n\}$ if $n \in \mathbb{N}$, and \mathbb{N} if $n = \infty$. We let $\bigcup_{i \in I_n} C_i$ denote $\bigcup_{i=0}^n C_i$.

Lemma A.1. *Let ϕ be a POCTL* formula, and $\{\sigma \in Path \mid \sigma \models \phi\}$ be a measurable set. Then, there exist pairwise disjoint basic cylinder sets $C_0, \dots, C_n \in \mathcal{C}$ such that*

$$\bigcup_{i \in I_n} C_i = \{\sigma \in Path \mid \sigma \models \phi\}$$

where $n \in \mathbb{N} \cup \{\infty\}$.

Proof. Follows directly from the definition of measurable set. □

Now we prove the theorem by structural induction over the path formula ϕ . For short, we write $path(\phi)$ to denote the set $\{\sigma \in Path \mid \sigma \models \phi\}$.

- $a, \neg\Phi, \Phi \wedge \Phi, \epsilon$: Therefore, ϕ is a state formula. We simply obtain

$$path(\phi) = \bigcup_{\substack{s \in S \\ s \models \phi}} \mathcal{C}(s)$$

- $\neg\phi$: $path(\neg\phi) = Path \setminus path(\phi)$. By induction hypothesis, the set $path(\phi)$ is measurable, thus, the complement is also measurable.
- $\phi \wedge \psi$: By induction hypothesis, $path(\phi)$ and $path(\psi)$ are measurable. By Lemma A.1, we have $path(\phi) = \bigcup_{i \in I_m} C_i$ and $path(\psi) = \bigcup_{i \in I_n} C'_i$, where m, n, C_i, C'_i are as indicated in the lemma. Therefore,

$$\begin{aligned} path(\phi \wedge \psi) &= \left(\bigcup_{i \in I_m} C_i \right) \cap \left(\bigcup_{j \in I_n} C'_j \right) \\ &= \bigcup_{i \in I_m} \left(C_i \cap \left(\bigcup_{j \in I_n} C'_j \right) \right) = \bigcup_{i \in I_m} \left(\bigcup_{j \in I_n} (C_i \cap C'_j) \right) \end{aligned}$$

by Lemma 3.4.2, $C_i \cap C'_j$ equals either C_i or C'_j or \emptyset . We prove further that this union can be simplified to a union of disjoint basic cylinder sets. Let us first assume that $C_i \cap C'_j$ and $C_{i'} \cap C'_{j'}$ are not disjoint. Then, there are only two possibilities, i.e., either $C_i \cap C'_j = C_i$, $C_{i'} \cap C'_{j'} = C'_{j'}$ or $C_i \cap C'_j = C'_j$, $C_{i'} \cap C'_{j'} = C_{i'}$. We consider only the first case, i.e., C_i and $C'_{j'}$ are not disjoint, then either $C_i \subseteq C'_{j'}$ or $C'_{j'} \subseteq C_i$, which implies that the union of the two set equals one of them. The second case follows the same proof line. Thus, $\bigcup_{i \in I_m} \left(\bigcup_{j \in I_n} (C_i \cap C'_j) \right)$ can be simplified to a union of disjoint basic cylinder sets.

- $\mathbf{X}_o \phi$: By induction hypothesis, we have $path(\phi) = \bigcup_{i \in I_n} C_i$, where n, C_i are as indicated in Lemma A.1. We observe

$$path(\mathbf{X}_o \phi) = \bigcup_{s \in S} \bigcup_{s' \in S} \{ \sigma \in Path \mid \sigma_s[0] = s \wedge \sigma_{s'}[0] = o \wedge \sigma_s[1] = s' \wedge \sigma[1] \models \phi \}$$

where the conjunction is measurable because

$$\{ \sigma \in Path \mid \sigma[1] \models \phi \} = \bigcup_{\substack{s \in S \\ \exists s' \in S. P(s', s) > 0}} \{ \sigma \in Path \mid \sigma_s[0] = s \wedge \sigma \models \phi \}$$

is measurable by induction hypothesis. Thus, the set $path(\mathbf{X}_o \phi)$ is measurable. Consider the probability measure \Pr_s , then,

$$\Pr_s(path(\mathbf{X}_o \phi)) = \mu_s(o) \sum_{s' \in S} P(s, s') \Pr_{s'}(path(\phi))$$

- $\phi \mathcal{U}^{\leq n} \psi$: We first define $\sigma \models \phi \mathcal{U}^i \psi$ by induction on i by $\sigma \models \phi \mathcal{U}^0 \psi$ iff $\sigma \models \psi$ and, for $i > 0$,

$$\sigma \models \phi \mathcal{U}^i \psi \text{ iff } \bigwedge_{j < i} \sigma[j] \models (\phi \wedge \neg \psi) \wedge \sigma[i] \models \psi$$

By definition, the sets $path(\phi \mathcal{U}^i \psi)$ are pairwise disjoint for all $i \in I_n$. Since

$$path(\phi \mathcal{U}^{\leq n} \psi) = \bigcup_{i \in I_n} path(\phi \mathcal{U}^i \psi)$$

$$path(\phi \mathcal{U}^i \psi) = \left\{ \sigma \in Path \mid \left(\bigwedge_{j < i} \sigma[j] \models \phi \wedge \neg \psi \right) \wedge \sigma[i] \models \psi \right\}$$

it is sufficient to show that for all $i \in I_n$ the sets $\{ \sigma \in Path \mid \sigma[j] \models \phi \wedge \neg \psi \}$ where $j < i$ and the set $\{ \sigma \in Path \mid \sigma[i] \models \psi \}$ are measurable. These sets can be also considered as the nested next operator, e.g., $\{ \sigma \in Path \mid \sigma[i] \models \psi \} = path(\mathbf{X}^i(\psi))$ where \mathbf{X}^i is i -time next operator. By induction hypothesis, this is measurable. \square

Appendix B

Proof of Theorem 5.3.2

Theorem 5.3.2. *Let Σ denote the alphabet $\mathcal{P}(AP) \times \Theta$. The generalized Büchi automaton $\mathcal{A}_\phi = (\Sigma, Q, L, \delta, Q_0, \mathcal{F})$ constructed for the OLTL formula ϕ accepts exactly those infinite words over Σ that satisfy ϕ .*

The proof we shall present is an adaption of the one from Gerth *et al.* [19]. The two directions of the theorem are proven in Lemma B.7 and Lemma B.8.

From now on we let the line numbers refer to the algorithm in Figure 5.4. Let Σ denote the alphabet $\mathcal{P}(AP) \times \Theta$ and let ϕ be an OLTL formula. Let $\mathcal{A}_\phi = (\Sigma, Q, L, \delta, Q_0, \mathcal{F})$ be the generalized Büchi automaton for ϕ . Let $\Delta(q)$ denote the value of $Old(q)$ at the point where the construction of the node q is finished, i.e., when it is added to $NodesSet$, at line 7. Let $\bigwedge \Xi$ denote the conjunction of a set of formulas Ξ , and let the conjunction of the empty set be equal to *tt*. For sets Ξ_1, Ξ_2 , $\bigwedge \Xi_1 \wedge \bigwedge \Xi_2$ is equivalent to $\bigwedge (\Xi_1 \cup \Xi_2)$. Let $\bigwedge_{\psi \in Next(q)} \mathbf{X}_{f(\psi, q)} \psi$ denote the conjunction of formulas $\mathbf{X}_{f(\psi, q)} \psi$ where $\psi \in Next(q)$ and $f(\psi, q)$ equals Ω if $\mathbf{X}_\Omega \psi \in Old(q)$ and Θ otherwise.

Recall that for an infinite sequence $w = (w_0, o_0), (w_1, o_1), \dots$ over $(\mathcal{P}(AP) \times \Theta)$ we let $w[i]$ denote the suffix of the sequence w starting with (w_i, o_i) . Similarly, for a run $\pi = q_0, q_1, \dots$ over \mathcal{A}_ϕ , $\pi[i]$ denotes the suffix of the run π starting with q_i . A run $\pi = q_0, q_1, \dots$ is called an execution iff $q_0 \in Q_0$.

Lemma B.1. *Let $\pi = q_0, q_1, \dots$ be an execution over \mathcal{A}_ϕ and let $\phi_1 \cup \phi_2 \in \Delta(q_0)$. Then, one of the following holds:*

1. $\forall i \geq 0. \phi_1, \phi_1 \cup \phi_2 \in \Delta(q_i)$ and $\phi_2 \notin \Delta(q_i)$.
2. $\exists j \geq 0. \forall 0 \leq i < j. \phi_1, \phi_1 \cup \phi_2 \in \Delta(q_i)$ and $\phi_2 \in \Delta(q_j)$.

Proof. Follows directly from the construction. □

Lemma B.2. *Suppose that the function $EXPAND(q, NodesSet)$ is called, and that in line 9, η is assigned one of $\phi_1 \cup \phi_2$ or $\phi_1 \vee \phi_2$ or $\phi_1 \wedge \phi_2$. The node q is split into two nodes $Node_1$ and $Node_2$ (lines 25–30). Immediately before the recursive call in line 31, the following holds:*

$$\chi := \bigwedge Old(q) \wedge \bigwedge New(q) \wedge \bigwedge_{\psi \in Next(q)} \mathbf{X}_{f(\psi, q)} \psi$$

is equivalent to

$$\chi_1 \vee \chi_2 := \left(\bigwedge Old(Node_1) \wedge \bigwedge New(Node_1) \wedge \bigwedge_{\psi \in Next(Node_1)} \mathbf{X}_{f(\psi, Node_1)} \psi \right) \vee \left(\bigwedge Old(Node_2) \wedge \bigwedge New(Node_2) \wedge \bigwedge_{\psi \in Next(Node_2)} \mathbf{X}_{f(\psi, Node_2)} \psi \right)$$

Proof. We only show the case that $\eta = \phi_1 \cup \phi_2$ (line 24). By the construction (lines 25–30), we have:

$$\begin{aligned} Old(Node_1) &= Old(q) \cup \{\eta\} \\ New(Node_1) &= (New(q) \setminus \{\eta\}) \cup (\{\phi_1\} \setminus Old(q)) = (New(q) \cup \{\phi_1\}) \setminus (\{\eta\} \cup Old(q)) \\ Next(Node_1) &= Next(q) \cup \{\eta\} \end{aligned}$$

Therefore,

$$\begin{aligned} \chi_1 &\iff \bigwedge Old(Node_1) \wedge \bigwedge New(Node_1) \wedge \bigwedge_{\psi \in Next(Node_1)} \mathbf{X}_{f(\psi, Node_1)} \psi \\ &\iff \bigwedge (Old(Node_1) \cup New(Node_1)) \wedge \bigwedge_{\psi \in Next(q) \cup \{\eta\}} \mathbf{X}_{f(\psi, Node_1)} \psi \\ &\iff \bigwedge (Old(q) \cup \{\eta\} \cup New(q) \cup \{\phi_1\}) \wedge \left(\bigwedge_{\psi \in Next(q)} \mathbf{X}_{f(\psi, q)} \psi \right) \wedge \mathbf{X}_{f(\eta, Node_1)} \eta \\ &\iff \chi \wedge (\eta \wedge \phi_1 \wedge \mathbf{X}\eta) \end{aligned} \tag{B.1}$$

Similarly, we get $\chi_2 \iff \chi \wedge (\eta \wedge \phi_2)$. Therefore,

$$\chi_1 \vee \chi_2 \iff \chi \wedge \eta \wedge ((\phi_1 \wedge \mathbf{X}\eta) \vee \phi_2)$$

where $(\phi_1 \wedge \mathbf{X}\eta) \vee \phi_2$ is equivalent to η . Since we have $\eta \in New(q)$, $\chi_1 \vee \chi_2$ is equivalent to χ . The other cases, i.e., $\eta = \phi_1 \cap \phi_2$ or $\eta = \phi_1 \vee \phi_2$, are treated similarly. \square

Lemma B.3. *Suppose that the function $EXPAND(q, NodesSet)$ is called. If the node q is updated to become a new node q' , as in lines 1–23, then,*

$$\chi := \bigwedge Old(q) \wedge \bigwedge New(q) \wedge \bigwedge_{\psi \in Next(q)} \mathbf{X}_{f(\psi, q)} \psi$$

is equivalent to

$$\chi' := \bigwedge Old(q') \wedge \bigwedge New(q') \wedge \bigwedge_{\psi \in Next(q')} \mathbf{X}_{f(\psi, q')} \psi$$

Proof. We consider all the possible positions:

- Line 3: Trivial, since $Old(q') = Old(q)$, $New(q') = New(q)$, $Next(q') = Next(q)$.
- Line 15: Trivial, since $Old(q') = Old(q) \cup \{\eta\}$, $New(q') = New(q) \setminus \{\eta\}$, $Next(q') = Next(q)$.

- Lines 17–20 ($\eta = \phi_1 \wedge \phi_2$): By the construction (lines 19–20), we have

$$\begin{aligned} Old(q') &= Old(q) \cup \{\eta\} \\ New(q') &= (New(q) \setminus \{\eta\}) \cup (\{\phi_1, \phi_2\} \setminus Old(q)) = (New(q) \cup \{\phi_1, \phi_2\}) \setminus (\{\eta\} \cup Old(q)) \\ Next(q') &= Next(q) \end{aligned}$$

Similar to Equation B.1, we get $\chi' = \chi \wedge \phi_1 \wedge \phi_2$. Since $\eta = \phi_1 \wedge \phi_2$ and $\eta \in New(q)$, χ' is equivalent to χ .

- Lines 21–23 ($\eta = \mathbf{X}_\Omega \phi'$): By the construction (lines 22–23), we have

$$Old(q') = Old(q) \cup \{\eta\}, New(q') = New(q) \setminus \{\eta\}, Next(q') = Next(q) \cup \{\phi'\}$$

Therefore,

$$\begin{aligned} \bigwedge_{\psi \in Next(q')} \mathbf{X}_{f(\psi, q')} \psi &= \bigwedge_{\psi \in Next(q) \cup \{\phi'\}} \mathbf{X}_{f(\psi, q')} \psi \\ &= \left(\bigwedge_{\psi \in Next(q)} \mathbf{X}_{f(\psi, q)} \psi \right) \wedge \mathbf{X}_{f(\phi', q')} \phi' \\ &= \bigwedge_{\psi \in Next(q)} \mathbf{X}_{f(\psi, q)} \psi \wedge \mathbf{X}_\Omega \phi' \end{aligned}$$

Similar to Equation B.1, we get $\chi' = \chi \wedge \mathbf{X}_\Omega \phi'$. Since $\eta = \mathbf{X}_\Omega \phi'$ and $\eta \in New(q)$, χ' is equivalent to χ . □

Using the field *Father* we can link each node to the one from which it was split. This defines an ancestor relation R over the graph nodes, where $(p, q) \in R$ iff $Father(q) = Name(p)$. Let R^* be the transitive closure of R . Nodes q , such that $Father(q) = Name(q)$, i.e., $(p, p) \in R$, are called rooted. A rooted node p can be either the initial node with $New(p) = \{\phi\}$, or obtained at lines 5–7 from some node q whose construction is finished. In the latter case, we have $New(p)$ set to $Next(q)$.

Lemma B.4. *Let p be a rooted node, and let q_1, q_2, \dots, q_n be all nodes, such that for all $1 \leq i \leq n$, $(p, q_i) \in R^*$, and $New(q_i) = \emptyset$, i.e., the construction of the node q_i is finished. Let Ξ be the set of formulas that are in $New(p)$, when it is created. Then, $\bigwedge \Xi$ is equivalent to*

$$\bigvee_{1 \leq i \leq n} \left(\bigwedge \Delta(q_i) \wedge \bigwedge_{\psi \in Next(q_i)} \mathbf{X}_{f(\psi, q_i)} \psi \right)$$

Moreover, if $w \models \bigvee_{1 \leq i \leq n} \left(\bigwedge \Delta(q_i) \wedge \bigwedge_{\psi \in Next(q_i)} \mathbf{X}_{f(\psi, q_i)} \psi \right)$, then there exists some $1 \leq i \leq n$ such that $w \models \bigwedge \Delta(q_i) \wedge \bigwedge_{\psi \in Next(q_i)} \mathbf{X}_{f(\psi, q_i)} \psi$ such that for each $\phi_1 \cup \phi_2 \in \Delta(q_i)$ with $w \models \phi_2$, ϕ_2 is also in $\Delta(q_i)$.

Proof. Follows by repeatedly using Lemma B.2 and Lemma B.3. Note that the construction of node q_i is finished, which implies that the field $New(q_i)$ is empty, therefore, $\bigwedge New(q_i) = tt$. □

Lemma B.5. *Let q be a node, whose construction is finished. Let $w = (w_0, o_0), (w_1, o_1), \dots$ with $w \models \bigwedge \Delta(q) \wedge \bigwedge_{\psi \in Next(q)} \mathbf{X}_{f(\psi, q)} \psi$. Then, there exists a transition $q \rightarrow q'$ in \mathcal{A}_ϕ such that $w[1] \models \bigwedge \Delta(q') \wedge \bigwedge_{\psi \in Next(q')} \mathbf{X}_{f(\psi, q')} \psi$. Moreover, let*

$$\Gamma = \{\phi_2 \mid \phi_1 \cup \phi_2 \in \Delta(q) \text{ and } \phi_2 \notin \Delta(q) \text{ and } w[1] \models \phi_2\}$$

Then, in particular there exists a transition $q \rightarrow q'$ such that q' also satisfies $\Gamma \subseteq \Delta(q')$.

Proof. When the construction of node q was finished, a rooted node r with $New(r) = Next(q) = \Xi$ was generated (line 6). The fact that $w \models \bigwedge_{\psi \in Next(q)} \mathbf{X}_{f(\psi, q)} \psi$ implies $o_0 \in \bigcap_{\psi \in Next(q)} f(\psi, q)$ and $w[1] \models \bigwedge Next(q) = \bigwedge \Xi$. Let q_1, \dots, q_n be all descendant nodes of r , applying Lemma B.4, we obtain:

$$w[1] \models \bigvee_{1 \leq i \leq n} \left(\bigwedge \Delta(q_i) \wedge \bigwedge_{\psi \in Next(q_i)} \mathbf{X}_{f(\psi, q_i)} \psi \right)$$

Moreover, there exists some $1 \leq i \leq n$ such that $w[1] \models \bigwedge \Delta(q_i) \wedge \bigwedge_{\psi \in Next(q_i)} \mathbf{X}_{f(\psi, q_i)} \psi$ such that for each $\phi_1 \cup \phi_2 \in \Delta(q_i)$ with $w[1] \models \phi_2$, ϕ_2 is also in $\Delta(q_i)$. For $\zeta' \in \Gamma$, we have $\zeta \cup \zeta' \in \Delta(q)$, $\zeta' \notin \Delta(q)$ and $w[1] \models \zeta'$. From $w \models \bigwedge \Delta(q)$ we obtain $w \models \zeta \cup \zeta'$ and $w \not\models \zeta'$ which implies that $w[1] \models \zeta \cup \zeta'$, thus, $\zeta \cup \zeta' \in \Delta(q_i)$. Together with $w[1] \models \zeta'$ we obtain $\zeta' \in \Delta(q_i)$, thus, $\Gamma \subseteq \Delta(q_i)$. The fact $q \rightarrow q_i$ follows directly from the construction. \square

Lemma B.6. *Let $\pi = q_0, q_1, \dots$ be a run. If π accepts $w = (w_0, o_0), (w_1, o_1), \dots$, then, $w \models \bigwedge \Delta(q_0)$.*

Proof. Recall π accepts w , iff π is an accepting execution such that $w[i] \in L(\pi[i])$. Let $\phi_0 \in \Delta(q_0)$. We show that, by structural induction over ϕ_0 , if π is an execution which accepts w , then, $w \models \phi_0$. Recall the labeling function of q_i is a pair $(L_1(q_i), L_2(q_i))$. The first component $L_1(q_i)$ is equal to

$$\{X \mid X \subseteq AP \wedge Pos(q_i) \subseteq X \wedge X \cap Neg(q_i) = \emptyset\}$$

where $Pos(q_i)$ is $\Delta(q_i) \cap AP$ and $Neg(q_i)$ is $\{\eta \in AP \mid \neg \eta \in \Delta(q_i)\}$, i.e., $Pos(q_i)$ and $Neg(q_i)$ are the positive and negative occurrences of the propositions in $\Delta(q_i)$, respectively. The second component $L_2(q_i)$ is equal to $Obser(q_i)$ which is the value of the field *Observations* for q_i , whose construction is finished. We have following cases of ϕ_0 :

- $a \in AP$: Since $a \in \Delta(q_0)$, we obtain $a \in l$ for all $l \in L_1(q_0)$ by the definition of the labeling function. π accepts w implies that $w_i \in L_1(q_i)$ for $i \geq 0$. In particular we obtain $w_0 \in L_1(q_0)$ which implies that $a \in w_0$, thus, $w \models a$.
- $\neg a$ where $a \in AP$: $\neg a \in \Delta(q_0)$ implies that $a \notin l$ for all $l \in L_1(q_0)$. $w_0 \in L_1(q_0)$ implies that $a \notin w_0$ and further $w \models \neg a$.
- $\phi_1 \wedge \phi_2$: By the construction we have $\phi_1 \in \Delta(q_0)$ and $\phi_2 \in \Delta(q_0)$. By induction hypothesis, if π accepts w , we obtain $w \models \phi_1$ and $w \models \phi_2$ which implies immediately $w \models \phi_1 \wedge \phi_2$.
- $\mathbf{X}_\Omega \psi$: By the construction, $\psi \in Next(q_0)$, thus, $\psi \in \Delta(q_1)$. By induction hypothesis, we have that, if $\pi[1]$ accepts $w[1]$, $w[1] \models \psi$. The fact that π accepts w also implies $o_i \in L_2(q_i) = Obser(q_i)$. By the construction, we know that $Obser(q_i)$ is the intersection of Ω' with $\mathbf{X}_{\Omega'} \phi_1 \in Old(q_i)$. Since $\mathbf{X}_\Omega \psi \in \Delta(q_i)$, we obtain $o_0 \in \Omega$. Thus, $w \models \mathbf{X}_\Omega \psi$.
- $\phi_1 \cup \phi_2$: If π accepts w , only the second case of Lemma B.1 is possible, i.e.,

$$\exists j \geq 0. \forall 0 \leq i < j. \phi_1, \phi_1 \cup \phi_2 \in \Delta(q_i) \text{ and } \phi_2 \in \Delta(q_j)$$

By induction hypothesis, if $\pi[j]$ accepts $w[j]$, $w[j] \models \phi_2$ and for each $0 \leq i < j$, if $\pi[i]$ accepts $w[i]$, $w[i] \models \phi_1$. Thus, by the semantics definition of OLTL, if π accepts w , $w \models \phi_1 \cup \phi_2$. \square

Lemma B.7. *Let $w = (w_0, o_0), (w_1, o_1), \dots$ be an infinite sequence over Σ . Let $\pi = q_0, q_1, \dots$ be an execution of \mathcal{A}_ϕ , which accepts w . Then, $w \models \phi$.*

Proof. By Lemma B.6, we get $w \models \bigwedge \Delta(q_0)$. From the construction, we have $\phi \in \Delta(q_{init})$ for all $q_{init} \in Q_0$. The fact that $q_0 \in Q_0$ implies that $\phi \in \Delta(q_0)$, which concludes the proof. \square

Lemma B.8. *Let $w = (w_0, o_0), (w_1, o_1), \dots$ with $w \models \phi$. Then, there exists an execution $\pi = q_0, q_1, \dots$ of \mathcal{A}_ϕ that accepts w .*

Proof. Let $p = (\text{name}, \text{name}, \{\text{init}\}, \{\phi\}, \emptyset, \emptyset, \Theta)$ be the rooted node constructed at the beginning of the algorithm (see lines 32–33). From the construction, the fields *Incoming* of the descendant nodes q of p also contain *init* which implies that q is a initial state. Since Ξ is initially $\{\phi\}$, applying Lemma B.4, we obtain that ϕ is equivalent to

$$\bigvee_{q \in Q_0} \left(\bigwedge \Delta(q) \wedge \bigwedge_{\psi \in \text{Next}(q)} \mathbf{X}_{f(\psi, q)} \psi \right)$$

Because of $w \models \phi$, there exists a node $q_0 \in Q_0$ such that $w \models \bigwedge \Delta(q_0) \wedge \bigwedge_{\psi \in \text{Next}(q_0)} \mathbf{X}_{f(\psi, q_0)} \psi$. Now, we construct the run π by repeatedly using Lemma B.5. Namely, if $w[i] \models \bigwedge \Delta(q_i) \wedge \bigwedge_{\psi \in \text{Next}(q_i)} \mathbf{X}_{f(\psi, q_i)} \psi$, then choose q_{i+1} to be a successor of q_i that satisfies $w[i+1] \models \bigwedge \Delta(q_{i+1}) \wedge \bigwedge_{\psi \in \text{Next}(q_{i+1})} \mathbf{X}_{f(\psi, q_{i+1})} \psi$ and furthermore, for every $\phi_1 \cup \phi_2 \in \Delta(q_i)$, if ϕ_2 holds in $w[i+1]$, then $\phi_1 \in \Delta(q_{i+1})$.

From Lemma B.1 we know that $\phi_1 \cup \phi_2$ will propagate to the successors of q_i unless ϕ_2 holds. $\phi_1 \cup \phi_2 \in \Delta(q_i)$ implies $w[i] \models \phi_1 \cup \phi_2$. By definition, there must be some minimal $j \geq i$ such that $w[j] \models \phi_2$, thus, $\phi_2 \in \Delta(q_j)$. Obviously, the constructed execution satisfies the acceptance condition. The proof that π accepts w is as follows. $w[i] \models \bigwedge \Delta(q_i)$ implies $w_i \in L_1(q_i)$ and $w[i] \models \bigwedge_{\psi \in \text{Next}(q_i)} \mathbf{X}_{f(\psi, q_i)} \psi$ implies $o_i \in \bigcap_{\psi \in \text{Next}(q_i)} f(\psi, q_i)$. By definition of $L_2(q_i)$, we have $o_i \in L_2(q_i)$, which concludes the proof. \square

Appendix C

Counterexample

In Section 5.4 (QOS Formulas) we have presented the QOS model checking algorithm which is an adaption of the one introduced by Iyer & Narasimha [23]. As we indicated at step 1 of the algorithm in Figure 5.9, they considered the BSCCs instead of SCCs in the product automaton. But to prove the correctness, they used the following lemma:

If a BSCC B in the DTMC is the projection of an SCC B' in the product automaton, B' is a BSCC.

We construct a counterexample of it. Let a be an atomic proposition. First, we use the algorithm presented in Section 5.3 to construct the Büchi automaton A_ϕ for the OLTL (actually LTL) formula $\phi = \Box(a \vee \mathbf{X}a)$:

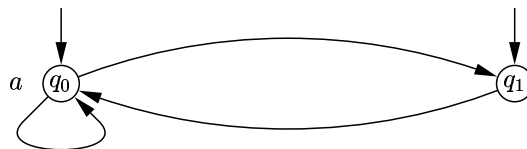


Figure C.1: The Büchi automaton for $\phi = \Box(a \vee \mathbf{X}a)$

The label of state q_0 is $\{\{a\}\}$ and of state q_1 is $\{\emptyset, \{a\}\}$. And both are initial states. Now we consider the following DTMC \mathcal{M} :

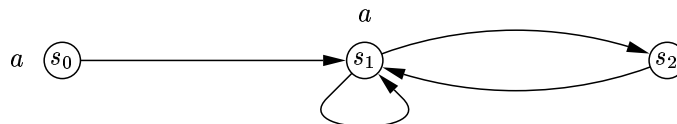
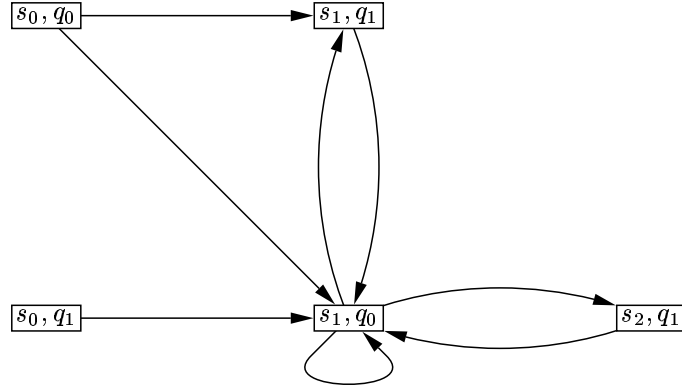


Figure C.2: A DTMC

The transition probability is given by: $\mathbf{P}(s_0, s_1) = \mathbf{P}(s_2, s_1) = 1$ and $\mathbf{P}(s_1, s_1) = \mathbf{P}(s_1, s_2) = \frac{1}{2}$. Applying the algorithm in Figure 5.11, we construct a part of the product automaton $\mathcal{M} \times A_\phi$ which starts with states (s_0, q_0) and (s_0, q_1) .

Figure C.3: The product automaton $\mathcal{M} \times A_\phi$

We observe that the set $S = \{(s_1, q_0), (s_2, q_1)\}$ is an SCC in the product automaton. Moreover, S projects to the BSCC $B = \{s_1, s_2\}$ onto \mathcal{M} . But obviously S is not a BSCC in the product automaton, instead, S is a proper subset of the BSCC $B' = \{(s_1, q_0), (s_2, q_1), (s_1, q_1)\}$. Moreover, S and B' project to the same BSCC B onto \mathcal{M} .

We also observe that the paths

$$\begin{aligned} &(s_0, q_0), (s_1, q_0), (s_1, q_0), (s_1, q_0), \dots \\ &(s_0, q_0), (s_1, q_1), (s_1, q_0), (s_1, q_0), \dots \end{aligned}$$

have the same projection s_0, s_1, s_1, \dots onto B . The reason is that the Büchi automaton is nondeterministic.

List of Figures

1.1	The Relations of the models	11
2.1	A DTMC	15
2.2	Expressiveness of the logics CTL*, CTL and LTL	16
2.3	Semantics of CTL*	16
2.4	Relationship of the logic PCTL* and its sublogics	17
2.5	Interpretation of LTL over $\mathcal{P}(AP)$	18
2.6	The algorithm for constructing a graph for an LTL formula	19
2.7	Splitting the until formula	20
2.8	The graph for $a \mathcal{U} \mathbf{X}b$	20
2.9	The generalized Büchi automaton for $a \mathcal{U} \mathbf{X}b$	21
2.10	CTL Model checking	24
3.1	Coin Toss Model with one fair coin and two biased coins	28
3.2	Updating belief states	29
3.3	A simple HMM	33
3.4	The induced DTMC of the HMM of Example 3.1.2 up to time 1	33
4.1	Semantics of POCTL*	39
4.2	Relationship of the logic POCTL* and its sublogics	40
5.1	Model checking algorithm for POCTL*	44
5.2	A simple hidden Markov model	44
5.3	Interpretation of OLTl over $\mathcal{P}(AP) \times \Theta$	46
5.4	The algorithm for constructing a graph for an OLTl formula	47
5.5	The graph for $a \mathcal{U} \mathbf{X}_o b$	48
5.6	The generalized Büchi automaton for $a \mathcal{U} \mathbf{X}_o b$	49
5.7	Splitting the bounded until formula	50
5.8	QOS Model checking	51
5.9	Model checking algorithm for QOS	52
5.10	The product automaton	53
5.11	Build the reachable part of the product automaton	55
C.1	The Büchi automaton for $\phi = \Box(a \vee \mathbf{X}a)$	67
C.2	A DTMC	67
C.3	The product automaton $\mathcal{M} \times A_\phi$	68

Bibliography

- [1] A. Aziz, V. Singhal, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. It Usually Works: The temporal Logic of stochastic systems. In P. Wolper, editor, *7th International Conference On Computer Aided Verification*, volume 939 of *Lecture Notes in Computer Science*, pages 155–165, Berlin, 1995. Springer Verlag.
- [2] S. Andova, H. Hermanns, and J.-P. Katoen. Discrete-time rewards model-checked. In K.G. Larsen and P. Niebert, editors, *Formal Modelling and Analysis of Timed Systems, FORMATS 2003*, Lecture Notes in Computer Science, Berlin, 2003. Springer Verlag.
- [3] C. Baier. On Algorithmic Verification Methods for Probabilistic Systems, 1998. Habilitationsschrift zur Erlangung der venia legendi der Fakultät für Mathematik and Informatik, Universität Mannheim.
- [4] C. Baier, E. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M Ryan. Symbolic Model Checking for Probabilistic Processes. In *Proceedings of the 24th International Colloquium On Automata Languages And Programming ICALP'97*, volume 1256 of *Lecture Notes in Computer Science*, pages 430–440, Berlin, 1997. Springer Verlag.
- [5] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. On the logical characterisation of performability properties. In *27th International Colloquium on Automata, Languages and Programming, ICALP 2000*, volume 1853 of *Lecture Notes in Computer Science*, pages 780–792, Berlin, July 2000. Springer Verlag.
- [6] C. Baier, B.R. Haverkort, H. Hermanns, and J.-P. Katoen. Efficient computation of time-bounded reachability probabilities in uniformized continuous-time Markov decision processes. 2003. Accepted for TACAS 2004.
- [7] C. Baier, B.R. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Transactions on Software Engineering*, 29(7):524–545, 2003.
- [8] C. Baier, H. Hermanns, J.-P. Katoen, and V. Wolf. Comparative Branching-Time Semantics for Markov Chains (extended abstract). In *CONCUR '03*, volume 2761 of *Lecture Notes in Computer Science*, pages 492–507, Berlin, September 2003. Springer Verlag.
- [9] G. Bhat, R. Cleaveland, and O. Grumberg. Efficient On-the-Fly Model Checking for CTL*. In *Proceedings of the Tenth Annual Symposium on Logic in Computer Science*, pages 388–397, California, June 1995. San Diego.
- [10] A. Bianco and L. de Alfaro. Model Checking of Probabilistic and Nondeterministic Systems. In *Foundations of Software Technology and Theoretical Computer Science (FST TCS 95)*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513, Berlin, 1995. Springer Verlag.

- [11] E. Birney. Hidden Markov models in biological sequence analysis. *IBM Journal of Research and Development*, 45(3/4):449–454, 2001.
- [12] E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL Model Checking. In D.L. Dill, editor, *Computer-Aided Verification, CAV '94*, volume 818 of *Lecture Notes in Computer Science*, pages 415–427, Berlin, June 1994. Springer Verlag.
- [13] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.
- [14] C. Courcoubetis and M. Yannakakis. Verifying Temporal Properties of Finite-State Probabilistic Programs. In *29th Annual Symposium on Foundations of Computer Science (FOCS '88)*, pages 338–345, Washington, USA, October 1988. IEEE Computer Society Press.
- [15] C. Courcoubetis and M. Yannakakis. The Complexity of Probabilistic Verification. *Journal of the ACM*, 42(4):857–907, 1995.
- [16] M. Daniele, F. Giunchiglia, and M. Y. Vardi. Improved automata generation for linear temporal logic. In N. Halbwachs and D. Peled, editors, *Eleventh Conference on Computer-Aided Verification (CAV 99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 249–260, Berlin, July 1999. Springer Verlag.
- [17] L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997. Technical report STAN-CS-TR-98-1601.
- [18] E.A. Emerson and C-L. Lei. Modalities for model checking: Branching time strikes back (Extended Abstract). In *Conference Record of the 12th Annual ACM Symposium on Principles of Programming Languages*, pages 84–96, New Orleans, January 1985. ACM Press.
- [19] R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper. Simple On-the-fly Automatic Verification of Linear Temporal Logic. In *Protocol Specification Testing and Verification*, pages 3–18, London, 1995. Chapman & Hall.
- [20] H. Hansson and B. Jonsson. A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
- [21] M. Hauskrecht. Value-Function Approximations for Partially Observable Markov Decision Processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.
- [22] H. Hermans. *Interactive Markov Chains: The Quest for Quantified Quality*, volume 2428 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, 2002.
- [23] P. Iyer and M. Narasimha. "Almost always" and "Definitely sometime" are not enough: Probabilistic quantifiers and Probabilistic model-checking. Technical Report TR-96-16, North Carolina State University, 1996.
- [24] D.N. Jansen. *Extensions of Statecharts with Probability, Time, and Stochastic Timing*. PhD thesis, University of Twente, Bern, Switzerland, October 2003. Inmarks ag.
- [25] D. Jurafsky and J.H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, Upper Saddle River, New Jersey, 2000.

- [26] J.-P. Katoen. Concepts, Algorithms, and Tools for Model Checking. Friedrich-Alexander Universität Erlangen-Nürnberg, Semester 1998/1999. Lecture Notes of the course "Mechanised Validation of Parallel Systems". <http://www.cs.auc.dk/~kgl/VERIFICATION99/katoen2.ps>.
- [27] J.G. Kemeny, J.L. Snell, and A.W. Knapp. *Denumerable Markov Chains*. Princeton, New Jersey, 1966.
- [28] D. Koller and X. Boyen. Tractable Inference for Complex Stochastic Processes. In *14th Annual Conference on Uncertainty in AI (UAI)*, pages 33–42, July 1998.
- [29] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer Verlag, Berlin, 1995.
- [30] R. D. Nicola and F. W. Vaandrager. Action versus state based logics for transition systems. In I. Guessarian, editor, *Semantics of Systems of Concurrent Processes*, volume 469 of *Lecture Notes in Computer Science*, pages 407–419, Berlin, 1990. Springer Verlag.
- [31] P. Poupart. Approximate Value-Directed Belief State Monitoring for Partially Observable Markov Decision Processes. Master's thesis, University of British Columbia, Vancouver, November 2000.
- [32] M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, New York, 1994.
- [33] L.R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.
- [34] J.G. Schmolze. *An Introduction to Hidden Markov Models*, Spring 2001. <http://www.cs.tufts.edu/comp/132/resources/hmm/>.
- [35] J.G. Schmolze. *Notes on Decision Theory and Artificial Intelligence*, October 2002. <http://www.cs.tufts.edu/comp/132/resources/mdp.pdf>.
- [36] G. Sigletos, G. Paliouras, and V. Karkaletsis. Role Identification from Free Text Using Hidden Markov Models. In *Second Conference of the Hellenic Association of Artificial Intelligence (SETN 02)*, pages 167–178, Thessaloniki, 2002.
- [37] F. Somenzi and R. Bloem. Efficient Büchi Automata from LTL Formulae. In *Twelfth Conference on Computer-Aided Verification (CAV 2000)*, volume 1633, pages 247–263, Berlin, 2000. Springer Verlag.
- [38] M. Y. Vardi and P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *Symposium on Logic in Computer Science (LICS '86)*, pages 332–345, Cambridge, Massachusetts, USA, June 1986. IEEE Computer Society Press.
- [39] J. A. Vlontzos and S. Y. Kung. Hidden Markov models for character recognition. *IEEE Transactions on Image Processing*, 1:539–543, October 1992.
- [40] P. Wolper. Constructing Automata from Temporal Logic Formulas: A Tutorial. In E. Brinksma, H. Hermanns, and J.-P. Katoen, editors, *Formal Methods and Performance Analysis (FMPA 2001)*, pages 261–277, Berlin, 2001. Springer Verlag.
- [41] P. Wolper, M. Y. Vardi, and A. P. Sistla. Reasoning about Infinite Computation Paths. In *24th Annual Symposium on Foundations of Computer Science (FOCS '83)*, pages 185–194, Los Angeles, USA, November 1982. IEEE Computer Society Press.

- [42] W. Zhang and N.L. Zhang. Solving Informative Partially Observable Markov Decision Processes. In *Proc. of the 6th European Conference on Planning (ECP)*, Lecture Notes in Computer Science Preprint, Berlin, 2001. Springer Verlag. <http://www2.cs.ust.hk/~wzhang/pub/ecp01.ps>.