AVACS – Automatic Verification and Analysis of Complex Systems

REPORTS

of SFB/TR 14 AVACS

Editors: Board of SFB/TR 14 AVACS

Logic and Model Checking for Hidden Markov Models

by
Lijun Zhang, Holger Hermanns, and David N. Jansen

AVACS Technical Report No. 6
July 2005
ISSN: 1860-9821

Publisher: Sonderforschungsbereich/Transregio 14 AVACS
(Automatic Verification and Analysis of Complex Systems)
Editors: Bernd Becker, Werner Damm, Martin Fränzle, Ernst-Rüdiger Olderog,
Andreas Podelski, Reinhard Wilhelm
ATRs (AVACS Technical Reports) are freely downloadable from `www.avacs.org`

Logic and Model Checking for Hidden Markov Models[⋆]

Lijun Zhang[1], Holger Hermanns[1,2], and David N. Jansen[2]

[1] Department of Computer Science, Saarland University, D-66123 Saarbrücken, Germany

[2]Department of Computer Science, University of Twente, Enschede, The Netherlands

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 HMMs 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 given sequence of observations" over HMMs. Subsequently, we give model checking algorithms for POCTL* over HMMs.

This report is the full version of a paper which appeared in FORTE'05 [25].

1 Introduction

Hidden Markov models (HMMs) [17] were developed in the late 1960's and have been proven to be very important for many applications, especially speech recognition [13], character recognition [22], biological sequence analysis [5], and protein classification problems [15]. Lately, HMMs receive increased attention in the context of communication channel modelling [20] and of QoS properties in wireless networks [9].

An HMM is a doubly embedded stochastic process with an underlying stochastic process over some state space, which is *hidden*. The occupied state 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

[⋆] Parts of this work was carried out while the third author was with the Max-Planck-Institut für Informatik, Saarbrücken. This work is partially supported by the NWO-DFG bilateral project Validation of Stochastic Systems (VOSS/VOSS-II), the NWO Vernieuwingsimpuls award 016.023.010 (VoPaD), and the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, see `www.avacs.org` for more information).

know the occupied state, but we do know the probability distribution over the set of states. This information is captured by a so-called belief state.

For a given HMM, one is often interested in the properties of the underlying stochastic process. In addition, one is also interested to reason about properties over the other set of stochastic processes which produce the observations. In this paper, we introduce a logic called POCTL*, which consists of state formulas, path formulas and belief state formulas. POCTL* allows us to specify properties of interests over HMMs. We consider the property:

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

This property can be expressed in POCTL* by $\mathcal{P}_{\geq 0.9}(\mathbf{X}_{o_0}\mathbf{X}_{o_1}\ldots\mathbf{X}_{o_n}tt)$. As indicated by Rabiner [17], this probability can be viewed as the score which specifies how well a given model matches the observations. In *Speech Recognition* [13], 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 O with probability at least 0.9, then we can almost conclude that this acoustic input represents the word "Need". In the protein classification problem, we want to classify the new protein to one known class. The idea is to construct an HMM for every known class, and calculate the score of the new protein under every class. The new protein belongs to the class which matches it (produces it with the highest probability).

On 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 also be considered as a variant of the temporal logic ACTL*, presented by De Nicola *et al.* [14], in which the usual next operator is extended to constrain the action label of the transition.

The PCTL* model checking [2, 1, 11] 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 [24, 21, 10], and then builds the product of the system and the constructed Büchi automaton. Finally, the QLS model checking problem can be reduced to a probabilistic reachability 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, where OLTL abbreviates Observational LTL) model checking problem. The latter can be further reduced to a probabilistic reachability 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.* [10].

2 Preliminaries

Rabin Automaton. A deterministic *Rabin automaton* [18, 2] is a tuple $\mathcal{R}_\phi = (\Sigma, Q, q_{in}, \delta, U)$ where Σ is a nonempty finite alphabet, Q is a finite set of states,

$q_{in} \in Q$ is the initial state, $\delta : Q \times \Sigma \to Q$ is the transition function, and $U = \{(P_i, R_i) \mid i = 1, \ldots, r\}$ is the Rabin acceptance condition where $P_i, R_i \subseteq Q$.

We call an infinite sequence $w = w_1, w_2, \ldots$ over Σ a *word* over Σ. w induces an unique path $\pi = q_0, q_1, \ldots$ in $\mathcal{R}$ where $q_0 = q_{in}$, and $q_{i+1} = \delta(q_i, w_i)$ for $i = 0, 1, \ldots$. π is an *accepting* path if

$$\inf(\pi) \subseteq P_j \text{ and } \inf(\pi) \cap R_j \neq \emptyset \text{ for some } j \in \{1, \ldots, r\}$$

where $\inf(\pi)$ denotes the set of states that occur infinitely often in π.

Discrete-time Markov Chains. A labeled discrete-time Markov chain (DTMC) is a tuple $\mathcal{D} = (S, \mathbf{P}, L)$ where S is a finite set of states, $\mathbf{P} : S \times S \to [0,1]$ is a probability matrix satisfying $\sum_{s' \in S} \mathbf{P}(s, s') \in \{0, 1\}$ for all $s \in S$, and $L : S \to 2^{AP}$ is a labeling function.

3 Hidden Markov Models

This section first recalls the concept of HMM, then defines belief states, paths over HMM, and probability spaces for a given HMM.

3.1 Labeled Discrete-Time HMMs

An HMM [17] 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 labeled DTMC:

Definition 1 *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, $\mu : S \times \Theta \longrightarrow [0,1]$ is an observation function satisfying $\sum_{o \in \Theta} \mu(s, o) = 1$ $\forall s \in S$, and α is an initial distribution on S such that $\sum_{s \in S} \alpha(s) = 1$.* $\square$

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. 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 labeled discrete-time HMM. For technical reasons, we assume there is no absorbing state in an HMM throughout our discussion[1].

[1] As indicated by Baier [2] (for concurrent probabilistic systems), this is a harmless restriction since any system can be transformed into an "equivalent" system without absorbing states. For an HMM $\mathcal{H}$ with absorbing states, we insert just a special state $\dagger$ with a self-loop and transitions from any absorbing state in $\mathcal{H}$ to $\dagger$.

3.2 Belief State

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, but, we can summarize the historical observations in a *belief state* (or *information state*) [12, 16] 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 is called the *belief space*, and is denoted by $\mathcal{B}$. We use S^t with $S^t \in S$ to denote the state at time t, and $O^t \in \Theta$ to denote the observation at time t. We write b_t to denote the belief state at time t.

Definition 2 *Let $o_i \in \Theta$ where $i = 0, \ldots, t$. The belief state b_t at time t, is the distribution over S at time t given the observation history $o_0, \ldots, o_t$:*

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

Now given the historical observations $o_0, \ldots, o_t$, the question is how to calculate the belief state b_n. 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 1.

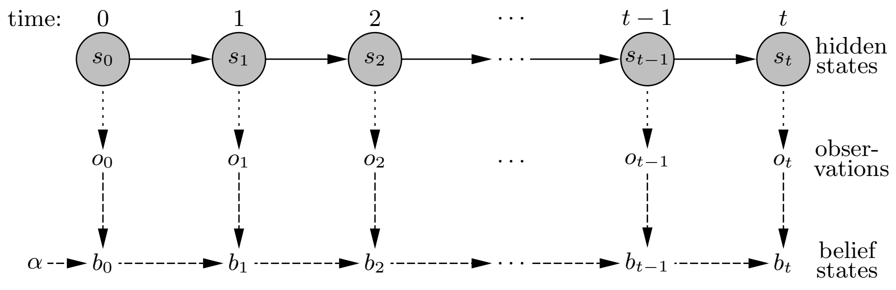

Fig. 1. Updating belief states

We depict the states in gray circles to indicate that they are hidden. The states together with the solid arrows between them represent the underlying state evolvement. 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. As a particular case, b_0 is a function of o_0 and the initial distribution α. Applying the Bayesian rule and the definition of b_0 we get: $b_0(s) = \frac{\alpha(s)\mu_s(o_0)}{K_0}$ where K_0 is a normalizing constant with value $\sum_{s \in S} \alpha(s)\mu_s(o_0)$.

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

$t = 1, \dots, n$. Again, applying the Bayesian rule and the definition of b_t we have: $b_{t+1}(s) = \frac{\sum_{s_t \in S} b_t(s_t)\mathbf{P}(s_t,s)\mu_s(o_{t+1})}{K_{t+1}}$ where K_{t+1} is a normalizing constant with value: $\sum_{s\in S}\left(\sum_{s_t\in S} b_t(s_t)\mathbf{P}(s_t,s)\mu_s(o_{t+1})\right)$. Hence, given the historical observations, we are able to calculate the current belief state.

3.3 Paths in HMM and Probability Spaces over Paths

Given $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$, let $s_i \in S$ and $o_i \in \Theta$ for all $i \in \mathbb{N}$. A path σ of $\mathcal{H}$ is a sequence $(s_0, o_0), (s_1, o_1) \ldots \in (S \times \Theta)^\omega$ 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 $S \times \Theta$.

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., $(s_i, o_i), (s_{i+1}, o_{i+1}), \ldots$. Note that $\sigma[0] = \sigma$.

Let $Path^{\mathcal{H}}$ denote the set of all paths in $\mathcal{H}$, and $Path^{\mathcal{H}}(s)$ denote the set of paths in $\mathcal{H}$ that start in s. The superscript $\mathcal{H}$ is ommitted whenever convenient. We define a probability space on paths of $\mathcal{H}$ using the standard cylinder construction. For a path $(s_0, o_0), (s_1, o_1), \ldots$, we define the *basic cylinder set* induced by the prefix of this path as follows:

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

If it is clear from the context, we use just $\mathcal{C}$ to denote this cylinder set. $\mathcal{C}$ consists of all paths σ starting with $(s_0, o_0), (s_1, o_1), \ldots (s_n, o_n)$. Let $\mathcal{C}yl$ contain all sets $\mathcal{C}((s_0, o_0), \ldots, (s_n, o_n))$ where $s_0, \ldots, s_n$ range over all state sequences and $o_0, \ldots, o_n$ range over all observation sequences. Let $\mathcal{F}$ be the σ-algebra on *Path* generated by $\mathcal{C}yl$. 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 measure[2] $\Pr_s$ 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 $n > 0$:

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

By induction on n, we obtain:

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

Lemma 3 *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 $\mathcal{C}yl$, and $\Pr_s$ is the probability measure which is described by Equation 1.* □

Let $b \in \mathcal{B}$ be a belief state, and $C \in \mathcal{C}yl$ 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, the triple $(Path, \mathcal{F}, \Pr_b)$ on domain *Path* is also a probability space.

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

4 The Logic POCTL*

This section 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 [2, 7, 6]. 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.* [14]. 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*

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:

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

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

The syntax of POCTL* consists of state formula, path formula and belief state formula. As in CTL*, we use $\varPhi, \varPsi$ 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 $O = (o_0, o_1, \ldots, o_n)$.

This can be expressed by a belief state formula $\epsilon = \mathcal{P}_{\geq 0.9}(\mathbf{X}_{o_0}\mathbf{X}_{o_1}\ldots\mathbf{X}_{o_n}tt)$. 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 $\mathbf{X}_{o_0}\mathbf{X}_{o_1}\ldots\mathbf{X}_{o_n}tt$ meets the bound ≥ 0.9. In *Speech Recognition* [13], 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 shall define later.

For the sake of simplicity, we do not consider the exist operator. The formula $\exists\phi$ is almost equivalent to 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 $\wedge, \Diamond, \Box$ which are defined in the same way as for CTL*. The timed variants of the temporal operators can be derived, e.g., $\Diamond^{\leq n}\phi = tt\,\mathcal{U}^{\leq n}\,\phi$, $\Box^{\leq n}\phi = \neg\Diamond^{\leq n}\neg\phi$.

4.2 Semantics of POCTL*

Let $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ be an HMM with $s \in S$ and $\sigma \in \mathit{Path}$. 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 $b \models \epsilon$.

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 2 where $\Pr_b\{\sigma \in \mathit{Path} \mid \sigma \models \phi\}$, or $\Pr_b(\phi)$ for short, denotes the probability measure of the set of all paths which satisfy ϕ and start states weighted by b.

$s \models a$	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	$\exists 0 \leq j \leq n.(\sigma[j] \models \psi \wedge \forall i < j.\sigma[i] \models \phi)$
$b \models \mathcal{P}_{\trianglelefteq p}(\phi)$	iff	$\Pr_b\{\sigma \in \mathit{Path} \mid \sigma \models \phi\} \trianglelefteq p$
$b \models \neg\epsilon$	iff	$b \not\models \epsilon$
$b \models \epsilon \wedge \epsilon'$	iff	$b \models \epsilon \wedge b \models \epsilon'$

Fig. 2. Semantics of POCTL*

A path satisfies the new operator $\mathbf{X}_o\phi$ if it starts with the observation o and the suffix[3] $\sigma[1]$ satisfies ϕ. Let Ω be a set of observations, i. e., $\Omega \subseteq \Theta$. We use the abbreviation $\mathbf{X}_\Omega\phi$ for $\bigvee_{o\in\Omega} \mathbf{X}_o\phi$ to shorten our notations.

By the definition of $\mathbf{X}_\Omega\phi$, we obviously have $\sigma \models \mathbf{X}_\Omega\phi$ iff $\sigma_o[0] \in \Omega \wedge \sigma[1] \models \phi$. 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*.

4.3 The Sublogics

An LTL formula together with a bound (QLS formula) can be interpreted over probabilistic models [2]. 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 that every next and until operator ($\mathbf{X}$,$\mathcal{U}^{\leq n}$) should be immediately enclosed in the probabilistic operator $\mathcal{P}$. The syntax of state and belief state formulas is the same as POCTL*, and the path formulas are given by:

$$\phi := \mathbf{X}_\Omega\Phi \mid \Phi\,\mathcal{U}^{\leq n}\,\Phi$$

where $\Omega \subseteq \Theta$.

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. Now we extend QLS to QOS (quantitative OLTL specification) which shall contribute to POCTL* model checking.

A QOS formula is a pair $(\phi, \trianglelefteq p)$ where ϕ is an OLTL formula, $\trianglelefteq \in \{\leq, <, \geq, >\}$ 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:

$$\mathcal{H}, s \models (\phi, \trianglelefteq p) \Longleftrightarrow \mathrm{Pr}_s(\phi) \trianglelefteq p$$

[3] This suffix $\sigma[1]$ is well-defined for we have previously assumed that the model does not contain any absorbing states.

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 3 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.

$$\begin{tikzcd}
\text{QOS} \arrow[r] & \text{POCTL}^* & \text{POCTL} \arrow[l] \\
\text{OLTL} \arrow[r] \arrow[u] & \text{OCTL}^* \arrow[u] & \text{OCTL} \arrow[l] \arrow[u]
\end{tikzcd}$$

Fig. 3. 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 following properties:

– The probability that the next observation is *head* and then the model goes to state *fair* meets the bound < 0.2.

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

This formula can be considered as a state formula or a belief state formula. A state (belief state) satisfies this formula if the probability calculated using the measure w. r. t. the state (belief state) meets the bound < 0.2.

– The probability is at most 0.05, that we eventually get an observation *head* and then move to state *fair*, whereas at any moment before we are either in state u_1 or state u_2.

$$\mathcal{P}_{\leq 0.05}((at_{u_1} \vee at_{u_2})\, \mathcal{U}\, \mathbf{X}_{head} at_{fair})$$

- With probability at least 0.9, the model generates the observation sequence $(o_0, o_1, \ldots, o_n)$.
$$\mathcal{P}_{\geq 0.9}(\mathbf{X}_{o_0}\mathbf{X}_{o_1}\ldots\mathbf{X}_{o_n}tt)$$
- The probability that the state sequence $(s_0, s_1, \ldots, s_n)$ produces the observation sequence $(o_0, o_1, \ldots, o_n)$ is at most 0.1.
$$\mathcal{P}_{\leq 0.1}(s_0 \wedge \mathbf{X}_{o_0}(s_1 \wedge \mathbf{X}_{o_1}(\ldots(s_n \wedge \mathbf{X}_{o_n}tt)\ldots)))$$
where s denotes the atomic proposition that the system is now in state s.

5 Model Checking

In this section, we present model checking algorithms for the logics POCTL*, POCTL and QOS. The model checking algorithm for POCTL* follows the same lines as the one for PCTL* [2, 7, 6]. It will first be reduced to the QOS model checking problem. The latter can further be reduced to a probabilistic reachability analysis. To that end, we construct a Büchi automaton for a given OLTL formula. The POCTL model checking algorithm can be adapted from the one presented by Hansson & Jonsson [11].

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* is an adaption of the one presented in [2] for PCTL*.

The algorithm is based on a recursive procedure that computes the sets $Sat(\Psi)$ for all state subformulas Ψ of Φ. The cases where Ψ is an atomic proposition or a negation or a conjunction is given by: $Sat(a) = \{s \in S \mid a \in L(s)\}$, $Sat(\neg\Psi_1) = S \backslash Sat(\Psi_1)$ and $Sat(\Psi_1 \wedge \Psi_2) = Sat(\Psi_1) \cap Sat(\Psi_2)$.

The case that Ψ is the probabilistic operator $\mathcal{P}_{\unlhd p}(\phi)$ is more involved. By the semantics, it is equivalent to check whether $\Pr_{b_s}(\phi)$ meets the bound $\unlhd\, p$, i. e., whether $\Pr_s(\phi) \unlhd p$. Let $\Psi_1, \ldots, \Psi_k$ be the maximal state subformulas of ϕ. The sets $Sat(\Psi_i)$ can be calculated recursively. Then, we replace $\Psi_1, \ldots, \Psi_k$ by the new atomic propositions $n_1, \ldots, n_k$ and extend the label of state s by n_i if $n_i \in Sat(\Psi_i)$.

We replace the subformulas $\Psi_1, \ldots, \Psi_k$ by new atomic propositions $n_1, \ldots, n_k$. The so obtained path formula ϕ' is an OLTL formula, and obviously we have $\Pr_s(\phi) = \Pr_s(\phi')$. Now we apply the QOS model checking algorithm to calculate $\Pr_s(\phi')$, which will be discussed in Section 5.3. Hence, the complexity of the POCTL* model checking algorithm is dominated by the one for QOS.

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 $\epsilon = \mathcal{P}_{\trianglelefteq p}(\phi)$ where ϕ is a POCTL* path formula. By definition,

$$b \models \mathcal{P}_{\trianglelefteq p}(\phi) \Longleftrightarrow p_b(\phi) \trianglelefteq p \Longleftrightarrow \sum_{s \in S} b(s)\mathrm{Pr}_s(\phi) \trianglelefteq p$$

therefore, it is sufficient to calculate $\mathrm{Pr}_s(\phi)$ for all $s \in S$.

5.2 POCTL Formulas

Let $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ 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 [11]. 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 for PCTL. Let $p \in [0,1]$, $\Omega \subseteq \Theta$ and $\trianglelefteq \in \{\leq, <, \geq, >\}$. We only need to consider the case that $\phi = \mathcal{P}_{\trianglelefteq p}(\mathbf{X}_\Omega \Phi')$. We observe that

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

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, $s \models \mathcal{P}_{\trianglelefteq p}(\mathbf{X}_\Omega \Phi')$ iff $p_s(\mathbf{X}_\Omega \Phi') \trianglelefteq p$.

5.3 QOS Formulas

This section presents the model checking algorithm for QOS formulas. We introduce two methods, an automaton based approach, which is based on the algorithm introduced by Baier *et al* [2, 4], and a direct method, where we reduce the problem to a PCTL* model checking problem over a DTMC, and apply the efficient algorithm presented by Courcoubetis *et al* [7].

An automaton based approach. The input is $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ with $s \in S$ and a QOS formula $(\phi, \trianglelefteq p)$ where $p \in [0, 1]$. We shall check whether $\mathcal{H}, s \models (\phi, \trianglelefteq p)$. In Appendix A, we present how to construct a Büchi automaton $\mathcal{A}_\phi$ for ϕ. This construction is an extension of the one presented by Gerth *et al.* [10]. By the result of Safra [18, 19], the Büchi automaton can be translated to a deterministic Rabin automaton. Let $\mathcal{R}_\phi = (\Sigma, Q, q_{in}, \delta, U)$ denote the Rabin automaton for ϕ. (Note that $\Sigma = \mathcal{P}(AP) \times \Theta$.) Next, we build the product automaton $\mathcal{H} \times \mathcal{R}_\phi$. Finally, the problem to calculate the measure of paths in $Path^{\mathcal{H}}(s)$ satisfying ϕ is reduced to a probabilistic reachability analysis in the product automaton. The method we shall present is an adaption of the one introduced by Bianco & de Alfaro [4], where we follow the presentation in [2].

The product automaton $\mathcal{H} \times \mathcal{R}_\phi = (S', \mathbf{P}', L')$ is given by: $S' = S \times Q$, $\mathbf{P}'((s, q), (s', q')) = \mathbf{P}(s, s') \cdot \mu_{s'}(o)$ if $q' \in \delta(q, (L(s'), o))$ and 0 otherwise.

For $s \in S$ and $o \in \Theta$, we define $s_R = (s, \delta(q_{in}, (L(s), o)))$. Let σ denote the path $(s_0, o_0), (s_1, o_1) \ldots$ in $\mathcal{H}$. Since R_ϕ is a deterministic automaton, we define the unique induced path $\sigma_R(s_0, q_0), (s_1, q_1), (s_2, q_2) \ldots$ in $\mathcal{H} \times R_\phi$, where $q_0 = \delta(q_{in}, (L(s_0), o_0))$, $q_{i+1} = \delta(q_i, (L(s_{i+1}), o_{i+1}))$.

Theorem 4 *Let $P'_i = S \times P_i$ and $R'_i = S \times R_i$. We define $U' = \cup_{1 \leq j \leq r} U'_j$, where U'_j is the largest subset of P'_j such that, for all $u' \in U'_j$: $reach^{\mathcal{H} \times R_\phi}(u') \subseteq U'_j$ and $reach^{\mathcal{H} \times R_\phi}(u') \cap R'_j \neq \emptyset$. Then,*

$$\mathrm{Pr}_s^{\mathcal{H}}(\phi) = \sum_{o \in \Theta} \mu_s(o) \cdot \mathrm{Pr}_{s_R}^{\mathcal{H} \times R_\phi}(reach(U'))$$

where $s_R = (s, \delta(q_{in}, (L(s), o)))$, and $\mathrm{Pr}_s^{\mathcal{H}}(\phi) = \mathrm{Pr}_s\{\sigma \in Path^{\mathcal{H}}(s) \mid \sigma \models \phi\}$ and $reach(U')$ denote the set of path which can reach U', i. e. $\{\sigma' \in Path^{\mathcal{H} \times \mathcal{R}_\phi}(s_R) \mid \exists i$ such that $\sigma'[i] \in U'\}$.

Proof. Let $\mathcal{C}((s, o_0), (s_1, o_1), \ldots, (s_n, o_n))$ be a basic cylinder set in $\mathcal{H}$ such that every path σ in $\mathcal{C}$ satisfies ϕ. The measure of $\mathcal{C}$ is $\mu_s(o_0) \prod_{i=1}^n \mathbf{P}(s_{i-1}, s_i)\mu_{s_i}(o_i)$. The induced unique cylinder set in $\mathcal{H} \times \mathcal{R}_\phi$ is $\mathcal{C}'((s, q_0), (s_1, q_1), \ldots, (s_n, q_n))$ where $q_0 = \delta(q_{in}, (L(s), o_0))$ and $q_{i+1} = \delta(q_i, (L(s_{i+1}), o_{i+1}))$ for $i = 1, \ldots, n$. Obviously, σ_R is in $\mathcal{C}'$. Since σ satisfies ϕ, the path $\pi = q_{in}, q_0, \ldots, q_n, \ldots$ must be an accepting path. Hence, there exists an i such that $\inf(\pi) \subseteq P_i$ and $\inf(\pi) \cap R_i \neq \emptyset$. By the definition of U', σ_R must contain at least one state which belongs to U'.

By construction of $\mathcal{H} \times \mathcal{R}_\phi$, the measure of $\mathcal{C}'$ is simply $\prod_{i=1}^n \mathbf{P}(s_{i-1}, s_i)\mu_{s_i}(o_i)$. Since $\mathcal{C}$ is an arbitrary cylinder set of interest, the above result is true for all $o_0 \in \Theta$. Let C_1, C_2 be two different cylinder sets in $\mathcal{H}$. Obviously, either one cylinder set includes another, or they are disjoint. Hence, summing up over all possible observations, we are done. □

Complexity. In Appendix A, we show that the Büchi automaton for the OLTL formula is exponential in the size of the formula. By the results of Safra [18, 19], the deterministic Rabin automaton for ϕ is double exponential in the size of the formula. So the overall complexity of the product automaton is linear in the size of the model, and double exponential in the size of the formula.

It thus remains to compute the reachability probability $\mathrm{Pr}_{s_R}^{\mathcal{H} \times R_\phi}(reach(U'))$ in the product automaton. To obtain this quantity, we can apply the method presented by de Alfaro [8, page 52]. The complexity is polynomial in the size of the product automaton.

A direct approach. The main idea of this approach is to construct a DTMC from the HMM, and transform the QOL formula ϕ to a QLS formula. Then, the original problem can be reduced to DTMC model checking problem.

We extend the set of atomic propositions by $AP' = AP \cup \{\Omega \mid \Omega \subseteq \Theta\}$. Given $\mathcal{H} = (S, \mathbf{P}, L, \Theta, \mu, \alpha)$ and a QOS formula $(\phi, \trianglelefteq p)$, we define the DTMC $\mathcal{D} = (S', \mathbf{P}', L')$ where $S' = S \times \Theta$, $\mathbf{P}'((s, o), (s', o')) = \mathbf{P}(s, s') \cdot \mu_{s'}(o')$ and

$L'(s,o) = L(s) \cup \{\Omega \subseteq \Theta \mid o \in \Omega\}$. Furthermore, we define a QLS formula $(\phi', \trianglelefteq p)$ as follows: Let $\mathbf{X}_\Omega \psi$ be a subformula of ϕ, we replace it by $\Omega \wedge \mathbf{X}\psi$, where Ω is a new atomic proposition. We proceed this process repeatedly until there is no next formula indexed with observations.

Lemma 5 $p_s^{\mathcal{H}}(\phi) = \sum_{o \in \Theta} \mu_s(o) \cdot p_{(s,o)}^{\mathcal{D}}(\phi')$

Proof. Similar to Lemma 4. □

Complexity. The constructed DTMC can be, in the worst case, $\mathcal{O}(|S|^2|\Theta|^2)$. We need still to calculate the probability measure of $\{\sigma \in \mathit{Path}^{\mathcal{D}} \mid \sigma \models \phi'\}$ in the DTMC. The optimal algorithm for that is given by Courcoubetis *et al* [7], and the complexity is polynomial in the size of the model, and exponential in the size of the formula.

In comparison to the other method, this method is single exponential in the size of the formula, but the DTMC suffers from the size $\mathcal{O}(|S|^2|\Theta|^2)$.

5.4 Improving the Efficiency

In this section, we discuss some efficiency issues for some special POCTL* formulas. After that we give some further improvements.

The Formula $s_0 \wedge \mathbf{X}_{o_0}(s_1 \wedge \mathbf{X}_{o_1}(\ldots(s_n \wedge \mathbf{X}_{o_n} tt)\ldots))$. For state $s \in S$, we let s denote also the atomic propositions which asserts that the model resides in state s. Given a basic cylinder set $\mathcal{C}((s_0,o_0),\ldots,(s_n,o_n))$, we define a formula $\phi = s_0 \wedge \mathbf{X}_{o_0}(s_1 \wedge \mathbf{X}_{o_1}(\ldots(s_n \wedge \mathbf{X}_{o_n} tt)\ldots))$ which is called the characteristic formula of this basic cylinder set. Obviously, $\{\sigma \in \mathit{Path} \mid \sigma \models \phi\} = \mathcal{C}((s_0,o_0),\ldots,(s_n,o_n))$. Hence, to check whether $s \models \mathcal{P}_{\trianglelefteq p}(\phi)$ boils down to checking whether the probability measure of the basic cylinder set, i. e., $\Pr_s(\mathcal{C})$, meets the bound $\trianglelefteq p$.

The Formula $\mathbf{X}_{o_0}\mathbf{X}_{o_1}\ldots\mathbf{X}_{o_n} tt$. We define a path formula $\phi = \mathbf{X}_{o_0}\mathbf{X}_{o_1}\ldots\mathbf{X}_{o_n} tt$ given the cylinder set $\mathcal{C}(o_0,\ldots,o_n) = \{\sigma \in \mathit{Path} \mid \forall i \leq n.\sigma_o[i] = o_i\}$. Obviously, $\{\sigma \in \mathit{Path} \mid \sigma \models \phi\} = \mathcal{C}(o_0,\ldots,o_n)$, which implies that to check whether $\alpha \models \mathcal{P}_{\trianglelefteq p}(\phi)$ boils down to checking whether $\sum_{s \in S} \alpha(s) \Pr_s(\mathcal{C})$ meets the bound $\trianglelefteq p$. The value $\Pr_s(\mathcal{C})$ can be calculated using Forward-Backward method presented in [17], with complexity $\mathcal{O}(|S|^2 n)$.

Building the Automaton by Need. 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., $\mathcal{P}_{\trianglelefteq p}(\phi)$ where there is no probabilistic operator in ϕ, we only need the states of the product automaton which are reachable from initial states s_R. So in this case we can construct the states of the product automaton as needed.

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 $(\phi\, \mathcal{U}\, \psi, \trianglelefteq p)$ (or $(\phi\, \mathcal{U}^{\leq n}\, \psi, \trianglelefteq p)$) where ϕ and ψ are POCTL* path formulas which can be verified recursively.

6 Conclusion and Future Work

6.1 Conclusion

In this paper, we have defined probability spaces (w. r. t. state and belief state) for a given HMM. 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. 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. Then, the QOS model checking problem is reduced to a probabilistic reachability analysis in the product automaton of the HMM and a deterministic Rabin 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. We plan to extend an HMM to a Hidden Markov decision process (HMDP) [4, 8] where 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, one first selects a probabilistic distribution over actions nondeterministically. Then, a successor can be chosen probabilistically according to the selected distribution over actions.

The nondeterminism is resolved by *schedulers* [3] (called *strategy* in [4, 8], *adversary* in [2]). 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 [4] $\Pr_s^\eta$ 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:

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

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 defined in a similar way. The model checking algorithm can be adapted from the one presented by de Alfaro for PCTL* formulas over MDPs.

HMDP with Fairness. Baier [2] 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.

Acknowledgements. The authors are grateful to Christel Baier (University of Bonn) and Frits Vaandrager (Radboud University Nijmegen) for helpful comments at an early state of the work presented in this paper.

References

1. Suzana Andova, Holger Hermanns, and Joost-Pieter Katoen. Discrete-time rewards model-checked. In *FORMATS, LNCS 2791:88-104*. Springer, 2003.
2. C. Baier. On Algorithmic Verification Methods for Probabilistic Systems, 1998. Habilitations- schrift zur Erlangung der venia legendi der Fakultät für Mathematik and Informatik, Universität Mannheim.
3. 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. In *TACAS, LNCS 2988:61-76*. Springer, 2004.
4. A. Bianco and L. de Alfaro. Model Checking of Probabilistic and Nondeterministic Systems. In *FSTTCS, LNCS 1026:499-513*. Springer, 1995.
5. E. Birney. Hidden Markov models in biological sequence analysis. *IBM Journal of Research and Development*, 45(3):449–454, 2001.
6. C. Courcoubetis and M. Yannakakis. Verifying Temporal Properties of Finite-State Probabilistic Programs. In *FOCS:338-345*. IEEE Computer Society Press, October 1988.
7. C. Courcoubetis and M. Yannakakis. The Complexity of Probabilistic Verification. *Journal of the ACM*, 42(4):857–907, 1995.
8. L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997. Technical report STAN-CS-TR-98-1601.
9. J.-M. François and G. Leduc. Mobility prediction's influence on QoS in wireless networks: A study on a call admission algorithm. In *3rd International Symposium on Modeling and Optimization in Mobile, Ad-Hoc and Wireless Networks*, pages 238–247. IEEE Computer Society, 2005.
10. R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper. Simple On-the-fly Automatic Verification of Linear Temporal Logic. In *PSTV 38:3-18*. Chapman & Hall, 1995.
11. H. Hansson and B. Jonsson. A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
12. M. Hauskrecht. Value-Function Approximations for Partially Observable Markov Decision Processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.
13. D. Jurafsky and J.H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2000.
14. R. D. Nicola and F. W. Vaandrager. Action versus state based logics for transition systems. In *Semantics of Systems of Concurrent Processes, LNCS 469:407-419*. Springer, 1990.
15. P.A. Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. The MIT Press, 2000.
16. P. Poupart. Approximate Value-Directed Belief State Monitoring for Partially Observable Markov Decision Processes. Master's thesis, University of British Columbia, November 2000.
17. 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.
18. S. Safra. On the complexity of ω-automata. In *FOCS*, pages 319–327, 1988.

19. S. Safra. Exponential determinization for ω-automata with strong-fairness acceptance condition. In *STOC*, pages 275–282, 1992.
20. K. Salamatian and S. Vaton. Hidden markov modeling for network communication channels. In *SIGMETRICS*, pages 92–101. ACM Press, 2001.
21. M. Y. Vardi and P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *LICS*, pages 332–345. IEEE Computer Society Press, June 1986.
22. J. A. Vlontzos and S. Y. Kung. Hidden Markov models for character recognition. *IEEE Transactions on Image Processing*, 1:539–543, October 1992.
23. P. Wolper. Constructing Automata from Temporal Logic Formulas: A Tutorial. In *FMPA 2001*, pages 261–277. Springer, 2001.
24. P. Wolper, M. Y. Vardi, and A. P. Sistla. Reasoning about Infinite Computation Paths. In *FOCS '83*, pages 185–194. IEEE Computer Society Press, 1982.
25. L. Zhang, H. Hermanns, and D.N. Jansen. Logic and Model Checking for Hidden Markov Chains. In *FORTE*. to appear.

A Constructing Büchi Automata from OLTL Formulas

In this appendix, we explain how to construct a Büchi automaton that accepts exactly all infinite sequences satisfying a given OLTL formula ϕ. This shall be used for QOS model checking in Section 5.3.

We assume that the given OLTL formula does not contain bounded until formulas. (OLTL formula containing bounded until formula can be dealt with similarly, but requires unwinding to a depth given by the (integer) bound decorating the operator.) Now we first give an interpretation of the OLTL formulas over the infinite words over the alphabet $\Sigma = \mathcal{P}(AP) \times \Theta$. Let $w = (w_0, o_0), (w_1, o_1), \ldots \in \Sigma^\omega$ be an infinite word. We write $w_1[i] = w_i$, $w_2[i] = o_i$ and $w[i]$ for the suffix of w starting with (w_i, o_i). The interpretation is given by:

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

The following construction is adapted from the above one for LTL formulas introduced by Gerth *et al.* [10]. 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.

A.1 Normal Form OLTL Formulas

A normal form OLTL formula ϕ is an OLTL 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:

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

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 OLTL formula ϕ can be iteratively transformed into a normal form OLTL formula using following rules:

$$
\begin{array}{ll}
\neg(\phi \vee \psi) \longrightarrow (\neg\phi) \wedge (\neg\psi) & \qquad \neg(\phi \wedge \psi) \longrightarrow (\neg\phi) \vee (\neg\psi) \\
\neg(\phi\, \mathcal{U}\, \psi) \longrightarrow \neg\phi\, \mathcal{V}\, \neg\psi & \qquad \neg(\phi\, \mathcal{V}\, \psi) \longrightarrow \neg\phi\, \mathcal{U}\, \neg\psi
\end{array}
$$

Additionally, the following lemma handles the operator $\mathbf{X}_\Omega \phi$ where $\Omega \subseteq \Theta$.

Lemma A1 *Let Ω be a subset of Θ, and ϕ be an OLTL formula. Then, $\neg\mathbf{X}_\Omega \phi \equiv \mathbf{X}_{\overline{\Omega}} tt \vee \mathbf{X}_\Omega \neg\phi$ 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,

$$\begin{aligned}\sigma \models \neg\mathbf{X}_\Omega\phi &\iff \neg(\sigma \models \mathbf{X}_\Omega\phi) \iff \neg(\sigma_o[0] \in \Omega \wedge \sigma[1] \models \phi)\\ &\iff \sigma_o[0] \notin \Omega \vee \sigma[1] \models \neg\phi \iff \sigma \models \mathbf{X}_{\overline{\Omega}}tt \vee \mathbf{X}\neg\phi\end{aligned}$$

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

We observe that the size of the resulting formula could be (in the worst case) $2|\phi|$, which is also the maximal blowup for $\mathbf{X}_\Omega$ formulas.

A.2 Creating Graphs

The algorithm is depicted in Figure 4. A graph node is a tuple

$$Node = (Name, Father, Incoming, New, Old, Next, Observations)$$

And we write $Name(Node) = Name$, $Father(Node) = Father$, $In(Node) = Incoming$, $New(Node) = New$, $Old(Node) = Old$, $Next(Node) = Next$, and $Obser(Node) = Observations$. The field $Name$ is a string that is the name of the node. 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 24–30). This field is used for reasoning about the correctness of the algorithm only, and is not important for the construction. The third field $Incoming$ 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 last field $Observations$ is a subset of Θ. This set contains all possible observations to satisfy the formulas in $Old(Node)$. The set $NodesSet$ contains all graph nodes whose construction is finished, i. e., the new field is empty.

For a given normal form OLTL formula ϕ_0, the procedure CreateGraph(ϕ_0) gives a set of nodes whose constructions are finished ($NodesSet$). Actually, the procedure Expand($Node, NodesSet$) is called where $Node$ equals $(\text{NAME}(), \text{NAME}(), \{init\}, \{\phi_0\}, \emptyset, \emptyset, \Theta)$ (line 32) 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 5–7.

```
$\textsc{Expand}(Node, NodesSet)$
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 $name = \textsc{Name}()$;
6              return $(\textsc{Expand}((name, name, Name(Node), Next(Node), \emptyset, \emptyset, \Theta),$
7                          $\{Node\} \cup NodesSet))$;
8  **else**
9      **let** $\eta \in New(Node)$;
10     $New(Node) := New(Node) \backslash \{\eta\}$;
11     **case** $\eta$ of
12     $\eta \in AP$ or $\eta = \neg\xi$ for some $\xi \in AP$ or $\eta = tt$ or $\eta = f\!f \Rightarrow$
13             **if** $\eta = f\!f$ 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 $(\textsc{Expand}(Node, NodesSet))$;
17     $\eta = \phi \wedge \psi \Rightarrow$
18             return $(\textsc{Expand}((Name(Node), Father(Node), In(Node),$
19                         $New(Node) \cup (\{\phi, \psi\} \backslash Old(Node)),$
20                         $Old(Node) \cup \{\eta\}, Next(Node), Obser(Node)), NodesSet))$;
21     $\eta = \mathbf{X}_{\Omega}\phi \Rightarrow$
22             return $(\textsc{Expand}((Name(Node), Father(Node), In(Node), New(Node),$
23                         $Old(Node) \cup \{\eta\}, Next(Node) \cup \{\phi\}, Obser(Node) \cap \Omega), NodesSet))$;
24     $\eta = \phi\, \mathcal{U}\, \psi$ or $\phi\, \mathcal{V}\, \psi$ or$\phi \vee \psi \Rightarrow$
25             $Node1 := (\textsc{Name}(), Father(Node), In(Node),$
26                         $New(Node) \cup (\{\textsc{New1}(\eta)\} \backslash Old(node)),$
27                         $Old(Node) \cup \{\eta\}, Next(Node) \cup \{\textsc{Next1}(\eta)\}, Obser(Node))$;
28             $Node2 := (\textsc{Name}(), Father(Node), In(Node),$
29                         $New(Node) \cup (\{\textsc{New2}(\eta)\} \backslash Old(node)),$
30                         $Old(Node) \cup \{\eta\}, Next(Node), Obser(Node))$;
31             return $(\textsc{Expand}(Node2, \textsc{Expand}(Node1, NodesSet)))$;

$\textsc{CreateGraph}(\phi)$
32 **let** $name = \textsc{Name}()$;
33 return $(\textsc{Expand}((name, name, \{init\}, \{\phi\}, \emptyset, \emptyset, \Theta), \emptyset))$;
```

Fig. 4. The algorithm for constructing a graph for an OLTL formula

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 negation of a proposition, either the current node is discarded (lines 13–14) or η is added to *Old* (lines 15–16). If η equals $\phi \wedge \psi$, both ϕ and ψ are added to *New* as the truth of both formulas is needed to make η hold (lines 17–20). If η is a next-formula, say $\mathbf{X}_{\Omega}\phi$, it suffices that ϕ holds at all immediate successors of *Node*. Thus, ϕ is added to *Next* (lines 21–23). Assuming that at the end of the construction the formulas $\mathbf{X}_{\Omega_1}\phi_1, \ldots, \mathbf{X}_{\Omega_k}\phi_k$ belong to $Old(Node)$. This implies that the formula $\mathbf{X}_{\Omega_1}\phi_1 \wedge \ldots \mathbf{X}_{\Omega_k}\phi_k$ is valid in *Node*. Then, *Observations* is equal to the set $\bigcap_{i\in\{1,\ldots,k\}} \Omega_i$ by observing that $\mathbf{X}_{\Omega_1}\phi_1 \wedge \mathbf{X}_{\Omega_2}\phi_2 \equiv \mathbf{X}_{\Omega_1\cap\Omega_2}\phi_1 \wedge \phi_2$.

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 24–31) and new formulas can be added to the fields *New* and *Next*. The function NAME() generates a new string for each call. The functions NEW1(η), NEW2(η) and NEXT1(η) are defined by Let $G = (V, E)$ be

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

a graph where V is the set of 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., $(p, q) \in E$.

A.3 The Generalized Büchi Automaton

A generalized Büchi automaton [10] is a tuple $\mathcal{A} = (\Sigma, Q, L, \delta, Q_0, \mathcal{F})$ where Σ is an alphabet, Q is a set of states, $L : Q \longrightarrow \mathcal{P}(\Sigma)$ is a labeling function, $\delta : Q \longrightarrow \mathcal{P}(Q)$ is a transition function, $Q_0 \subseteq Q$ is a set of initial states and $\mathcal{F} \subseteq \mathcal{P}(Q)$ is a set of accepting state sets.

A run π of $\mathcal{A}$ is an infinite sequence $q_0, q_1, \ldots \in Q^\omega$ such that $q_{i+1} \in \delta(q_i)$ for all $i \in \mathbb{N}$. A run $\pi = q_0, q_1, \ldots$ 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 $inf(\pi) \cap F \neq \emptyset$ for all $F \in \mathcal{F}$. Let $\pi[i]$ denote the suffix of the run π starting with q_i. An infinite word $w = w_0, w_1, \ldots \in \Sigma^\omega$ is accepted by the automaton A if there is an accepting execution $\pi = q_0, q_1, \ldots$ 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 OLTL formula ϕ as follows. The alphabet Σ is $\mathcal{P}(AP) \times \Theta$. 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 $init \in In(q)$. We have a transition $p \longrightarrow q$ if $(p, q) \in E$, i. e., if $p \in In(q)$.

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 second component $L_2(q)$ equals the set $\mathit{Obser}(q)$, i. e., the set of all possible observations to satisfy the formulas in $\mathit{Old}(q)$. The first component $L_1(q)$ contains all sets in $\mathcal{P}(AP)$ that are compatible with $\mathit{Old}(q)$. More precisely, let $\mathit{Pos}(q)$ be $\mathit{Old}(q) \cap AP$ and $\mathit{Neg}(q)$ be $\{\eta \in AP \mid \neg\eta \in \mathit{Old}(q)\}$, i. e., $\mathit{Pos}(q)$ and $\mathit{Neg}(q)$ are the positive and negative occurrences of atomic propositions in q, respectively. Then, the label of state q is defined by:

$$L_1(q) = \{X \mid X \subseteq AP \wedge \mathit{Pos}(q) \subseteq X \wedge X \cap \mathit{Neg}(q) = \emptyset\}$$

For each subformula of ϕ of the type $\psi_1 \,\mathcal{U}\, \psi_2$, we define a set $F \subseteq Q$ which includes the states $q \in Q$ such that either $\psi_1 \,\mathcal{U}\, \psi_2 \notin \mathit{Old}(q)$, or $\psi_2 \in \mathit{Old}(q)$. The construction of acceptance sets avoids accepting a run $q_0, q_1, \ldots$ in which $\psi_1 \,\mathcal{U}\, \psi_2$ appears from some node q_i onwards without ψ_2 occurring later. Let $\mathcal{F}$ consist of all F defined in this way. Obviously, we have $|\mathcal{F}| \leq |\phi|$.

The following theorem establishes the correspondence between OLTL formulas and generalized Büchi automata (proof see Appendix B).

Theorem A2 *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.* □

A.4 The Büchi Automaton

A Büchi automaton [23] $\mathcal{A}$ 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., $\mathit{inf}(\pi) \cap F \neq \emptyset$. The definitions of run, execution, accepting execution that were introduced for generalized Büchi automata carry over to Büchi automata in the obvious way.

For a generalized Büchi automaton $\mathcal{A}_\phi$, one can construct an equivalent Büchi automaton [23] with size $|\mathcal{F}||\mathcal{A}_\phi|$, where $|\mathcal{F}| \leq |\phi|$.

Example A3 The Büchi automaton $\mathcal{A}_\phi$ for the OLTL formula $\phi = a \,\mathcal{U}\, \mathbf{X}_o b$ is depicted in Figure 5. 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. $L_2(q)$ 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., $\mathcal{F} = \{q_1, q_2, q_3\}$. □

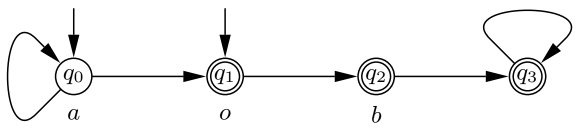

Fig. 5. The generalized Büchi automaton for $a\,\mathcal{U}\,\mathbf{X}_o b$

B Proof of Theorem A2

The proof we shall present is an adaption of the one from Gerth *et al.* [10]. The two directions of the theorem are proven in Lemma B7 and Lemma B8.

From now on we let the line numbers refer to the algorithm in Figure 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 $w = (w_0, o_0), (w_1, o_1), \ldots$ 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 $\pi = q_0, q_1, \ldots$ over $\mathcal{A}_\phi$, $\pi[i]$ denotes the suffix of the run π starting with q_i. A run $\pi = q_0, q_1, \ldots$ is called an execution iff $q_0 \in Q_0$.

Lemma B1 *Let $\pi = q_0, q_1, \ldots$ be an execution over $\mathcal{A}_\phi$ and let $\phi_1 \mathcal{U} \phi_2 \in \Delta(q_0)$. Then, one of the following holds:*

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

Proof. Follows directly from the construction. □

Lemma B2 *Suppose that the function* Expand$(q, NodesSet)$ *is called, and that in line 9, η is assigned one of $\phi_1 \,\mathcal{U}\, \phi_2$ or $\phi_1 \,\mathcal{V}\, \phi_2$ or $\phi_1 \vee \phi_2$. 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 $\eta = \phi_1 \,\mathcal{U}\, \phi_2$ (line 24). By the construction (lines 25–30), we have:

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

Therefore,

$$\begin{aligned}
\chi_1 &\iff \bigwedge Old(Node_1) \wedge \bigwedge New(Node_1) \wedge \bigwedge_{\psi \in Next(Node_1)} \mathbf{X}_{f(\psi, Node_1)} \psi \\
&\iff \bigwedge (Old(Node_1) \cup New(Node_1)) \wedge \bigwedge_{\psi \in Next(q) \cup \{\eta\}} \mathbf{X}_{f(\psi, Node_1)} \psi \\
&\iff \bigwedge (Old(q) \cup \{\eta\} \cup New(q) \cup \{\phi_1\}) \wedge \left(\bigwedge_{\psi \in Next(q)} \mathbf{X}_{f(\psi, q)} \psi \right) \wedge \mathbf{X}_{f(\eta, Node_1)} \eta \\
&\iff \chi \wedge (\eta \wedge \phi_1 \wedge \mathbf{X}\eta)
\end{aligned} \tag{2}$$

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., $\eta = \phi_1 \,\mathcal{V}\, \phi_2$ or $\eta = \phi_1 \vee \phi_2$, are treated similarly. □

Lemma B3 *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

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

.

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

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

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

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

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 B4 *Let p be a rooted node, and let $q_1, q_2, \ldots, q_n$ be all nodes, such that for all $1 \le i \le n$, $(p, q_i) \in R^*$, 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*

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

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

Proof. Follows by repeatedly using Lemma B2 and Lemma B3. 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 B5 *Let q be a node, whose construction is finished. Let $w = (w_0, o_0), (w_1, o_1), \ldots$ with $w \models \bigwedge \Delta(q) \wedge \bigwedge_{\psi \in Next(q)} \mathbf{X}_{f(\psi,q)}\psi$. Then, there exists a transition $q \longrightarrow q'$ in $\mathcal{A}_\phi$ such that*

$$w[1] \models \bigwedge \Delta(q') \wedge \bigwedge_{\psi \in Next(q')} \mathbf{X}_{f(\psi,q')}\psi$$

Moreover, let

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

Then, in particular there exists a transition $q \longrightarrow q'$ 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 $q_1, \ldots, q_n$ be all descendant nodes of r, applying Lemma B4, we obtain:

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

Moreover, there exists some $1 \le i \le n$ 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 $\phi_1 \,\mathcal{U}\, \phi_2 \in \Delta(q_i)$ with $w[1] \models \phi_2$, ϕ_2 is also in $\Delta(q_i)$. For $\zeta' \in \Gamma$, we have $\zeta \,\mathcal{U}\, \zeta' \in \Delta(q)$, $\zeta' \notin \Delta(q)$ and $w[1] \models \zeta'$. From $w \models \bigwedge \Delta(q)$ we obtain $w \models \zeta \,\mathcal{U}\, \zeta'$ and $w \not\models \zeta'$ which implies that $w[1] \models \zeta \,\mathcal{U}\, \zeta'$, thus, $\zeta \,\mathcal{U}\, \zeta' \in \Delta(q_i)$. Together with $w[1] \models \zeta'$ we obtain $\zeta' \in \Delta(q_i)$, thus, $\Gamma \subseteq \Delta(q_i)$. The fact $q \longrightarrow q_i$ follows directly from the construction. □

Lemma B6 *Let $\pi = q_0, q_1, \ldots$ be a run. If π accepts $w = (w_0, o_0), (w_1, o_1), \ldots$, 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 $l \in L(q_0)$. $w_0 \in L_1(q_0)$ 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, $w \models \mathbf{X}_\Omega \psi$.
- $\phi_1 \,\mathcal{U}\, \phi_2$: If π accepts w, only the second case of Lemma B1 is possible, i. e.,

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

  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, $w \models \phi_1 \,\mathcal{U}\, \phi_2$.

□

Lemma B7 *Let $w = (w_0, o_0), (w_1, o_1), \ldots$ be an infinite sequence over Σ. Let $\pi = q_0, q_1, \ldots$ be an execution of $\mathcal{A}_\phi$, which accepts w. Then, $w \models \phi$.*

Proof. By Lemma B6, 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. □

Lemma B8 *Let $w = (w_0, o_0), (w_1, o_1), \ldots$ with $w \models \phi$. Then, there exists an execution $\pi = q_0, q_1, \ldots$ of $\mathcal{A}_\phi$ that accepts w.*

Proof. Let $p = (name, name, \{init\}, \{\phi\}, \emptyset, \emptyset, \Theta)$ 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 B4, we obtain that ϕ is equivalent to

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

Because of $w \models \phi$, there exists a node $q_0 \in Q_0$ such that

$$w \models \bigwedge \Delta(q_0) \wedge \bigwedge_{\psi \in Next(q_0)} \mathbf{X}_{f(\psi,q_0)} \psi$$

Now, we construct the run π by repeatedly using Lemma B5. Namely, if $w[i] \models \bigwedge \Delta(q_i) \wedge \bigwedge_{\psi \in Next(q_i)} \mathbf{X}_{f(\psi,q_i)}\psi$, then choose q_{i+1} to be a successor of q_i that satisfies

$$w[i+1] \models \bigwedge \Delta(q_{i+1}) \wedge \bigwedge_{\psi \in Next(q_{i+1})} \mathbf{X}_{f(\psi,q_{i+1})}\psi$$

and furthermore, for every $\phi_1 \,\mathcal{U}\, \phi_2 \in \Delta(q_i)$, if ϕ_2 holds in $w[i+1]$, then $\phi_2 \in \Delta(q_{i+1})$.

From Lemma B1 we know that $\phi_1 \,\mathcal{U}\, \phi_2$ will propagate to the successors of q_i unless ϕ_2 holds. $\phi_1 \,\mathcal{U}\, \phi_2 \in \Delta(q_i)$ implies $w[i] \models \phi_1 \,\mathcal{U}\, \phi_2$. 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 $w[i] \models \bigwedge_{\psi \in Next(q_i)} \mathbf{X}_{f(\psi,q_i)}\psi$ implies $o_i \in \bigcap_{\psi \in Next(q_i)} f(\psi, q_i)$. By definition of $L_2(q_i)$, we have $o_i \in L_2(q_i)$, which concludes the proof. $\square$