Marimba: A Tool for Verifying Properties of Hidden Markov Models*

Noé Hernández[1], Kerstin Eder[3,4], Evgeni Magid[3,4], Jesús Savage[2], and David A. Rosenblueth[1]

[1] Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas
[2] Facultad de Ingeniería
Universidad Nacional Autónoma de México, D.F., México
no_hernan@ciencias.unam.mx, drosenbl@unam.mx, savage@servidor.unam.mx
[3] Department of Computer Science, University of Bristol, Bristol, BS8 1UB, UK
[4] Bristol Robotics Laboratory, Bristol, BS16 1QY, UK
Kerstin.Eder@bristol.ac.uk, Evgeni.Magid@bristol.ac.uk

Abstract. The formal verification of properties of Hidden Markov Models (HMMs) is highly desirable for gaining confidence in the correctness of the model and the corresponding system. A significant step towards HMM verification was the development by Zhang et al. of a family of logics for verifying HMMs, called POCTL*, and its model checking algorithm. As far as we know, the verification tool we present here is the first one based on Zhang et al.'s approach. As an example of its effective application, we verify properties of a handover task in the context of human-robot interaction. Our tool was implemented in Haskell, and the experimental evaluation was performed using the humanoid robot Bert2.

1 Introduction

A Hidden Markov Model (HMM) is an extension of a Discrete Time Markov Chain (DTMC) where the states of the model are hidden but the observations are visible. Typically, an HMM is studied with respect to the three basic problems examined by Rabiner in [9]. However, to the best of our knowledge, no practical model checker exists for HMMs despite their broad range of applications, e.g., speech recognition, DNA sequence analysis, text recognition and robot control. We describe in this paper a tool for verifying HMM properties written in the Probabilistic Observation Computational Tree Logic* (POCTL* [11]), and use this tool for verifying properties of a robot-to-human handover interaction.

POCTL* is a specification language for HMM properties. It is a probabilistic version of CTL* where a set of observations is attached to the *next* operator. Zhang et al. [11] sketched two model checking algorithms for POCTL*, an "automaton based" approach, and a "direct" approach. We opted for the direct

* The final version of this paper was accepted in the 13th International Symposium on Automated Technology for Verification and Analysis (ATVA 2015). The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-24953-7_14

approach for its lower time complexity. Noticeably, this approach produces a DTMC $\mathcal{D}$ and a Linear Temporal Logic (LTL) formula ϕ, so the PRISM [6] model checker could be used to verify this property. Such a model checker follows the automata based approach whose complexity is doubly exponential in $|\phi|$ and polynomial in $|\mathcal{D}|$, whereas we implemented the direct method by Courcoubetis et al. [1] whose complexity is singly exponential in $|\phi|$ and polynomial in $|\mathcal{D}|$, which is also the final complexity of our tool. This direct method repeatedly constructs a DTMC and rewrites an LTL formula, such that one temporal operator is removed each time while preserving the probability of satisfaction.

We have named our model checker Marimba. A marimba is a xylophone-like musical instrument that is popular in south-east Mexico and Central America. Marimba [5] was implemented in Haskell and compiled with GHCi. Our tool is available for download from `https://github.com/nohernan/Marimba`.

2 Tool architecture and implementation

Haskell was chosen to code this first version of Marimba since it allows us to work in a high-level abstract layer, by providing useful mechanisms like lazy evaluation and a pure functional paradigm. Furthermore, Haskell manages recursion efficiently; this is a valuable aspect because recursive calls are made continuously throughout the execution. As a future work, we consider coding Marimba in a language like Java and make it a symbolic model checker.

Marimba features a command-line interface. Furthermore, instead of working with a command window, a more user friendly and preferable execution is accomplished through the *Emacs* text editor extended with the `Haskell-mode`.

2.1 Marimba's input and modules

The first input is a `.poctl` file with the six elements of an HMM $\mathcal{H}$, namely a finite set of states S, a state transition probability matrix A, a finite set of observations Θ, an observation probability matrix B, a function L that maps states to sets of atomic propositions from a set $AP_{\mathcal{H}}$, and an initial probability distribution π over S. The second input is a POCTL* state formula Φ typed in the command window according to the syntactic rules:

$$\Phi ::= \mathsf{true} \mid \mathsf{false} \mid a \mid (\neg\Phi) \mid (\Phi \vee \Phi) \mid (\Phi \wedge \Phi) \mid (\mathcal{P}_{\bowtie p}(\phi)),$$
$$\phi ::= \Phi \mid (\neg\phi) \mid (\phi \vee \phi) \mid (\phi \wedge \phi) \mid (\mathbf{X}_{\mathsf{o}}\phi) \mid (\phi \mathcal{U}^{\leq n} \phi) \mid (\phi \mathcal{U} \phi),$$

where $a \in AP_{\mathcal{H}}$, $\mathsf{o} \in \Theta$, $n \in \mathbb{N}$, $p \in [0,1]$, and $\bowtie \in \{\leq, <, \geq, >\}$. In addition, we define $\mathbf{X}_{\Omega}\phi$ as a shorthand for $\bigvee_{\mathsf{o} \in \Omega} \mathbf{X}_{\mathsf{o}}\phi$ provided $\Omega \subseteq \Theta$. We examine below the six Haskell modules that constitute Marimba.

ModelChecker.hs performs the initial computations of the model checker for POCTL*. It recursively finds a most nested state subformula of Φ, not being a propositional variable, and the states of $\mathcal{H}$ that satisfy it. On the one hand, finding the states satisfying a propositional subformula is straightforward. On

the other hand, we invoke the module *DirectApproach.hs* to obtain the states satisfying a probabilistic state subformula. Next, this module extends the labels of such states with a new atomic proposition a. In Φ, the state subformula being addressed is replaced by a. The base case occurs when we reach a propositional variable, so we return the states that have it in their label.

DirectApproach.hs transforms the HMM $\mathcal{H}$ into a DTMC $\mathcal{D}$, and removes from the specification the observation set attached to the *next* operator $\mathbf{X}$ by generating a conjunction of the observation-free $\mathbf{X}$ with a new propositional variable. Thus, we obtain an LTL formula that is passed, together with $\mathcal{D}$, to the module *Courcoubetis.hs*. The new propositional variables are drawn from the power set of observations. Remarkably, it is not necessary to compute such a power set since the label of a state in $\mathcal{D}$ is easily calculated.

Courcoubetis.hs implements a modified version of the method by Courcoubetis et al. to find the probability that an LTL formula is satisfied in a DTMC. In this module, when dealing with the $\mathcal{U}$ and $\mathcal{U}^{\leq n}$ operators, we apply ideas from [10] for computing a partition of states of $\mathcal{D}$. Moreover, to handle the $\mathcal{U}$ operator we have to solve a linear equation system. To that end, we use the *linearEqSolver* library [3], which in turn executes the *Z3* theorem prover [2].

Lexer.hs and *Parser.hs* are in charge of the syntactic analysis of the input. Finally, *Main.hs* is loaded to start Marimba. This module manages the interaction with the user, and starts the computation by passing control to *ModelChecker.hs*.

In a typical execution, Marimba prompts the user to enter a `.poctl` file path. Next, our tool asks whether or not the user wants to take into account the initial distribution in the computation of the probability of satisfaction. This choice corresponds to opposite ideas presented in [1] and [11], i.e., the method by Courcoubetis et al. uses the initial distribution to define their probability measure, contrary to that defined by Zhang et al. Afterwards, a POCTL* formula has to be entered. Marimba returns the list of states satisfying this formula, and asks the user whether there are more formulas to be verified on the same model.

The `.poctl` file is simply a text file where the elements of an HMM are defined, e.g., the set of states is defined by the reserved word `States`, and if the model consists of five states, we write `States=5`. Likewise, POCTL* formulas have a natural writing, for example, $\mathcal{P}_{<0.1}(\mathbf{X}_{\{o_1\}}a)$ is typed as `P[<0.1](X_{1}a)`.

3 Verification of a human-robot interaction

We applied Marimba to a real-world example, namely the verification of the robot-to-human handover task [4] using the robot Bert2 [7] at the Bristol Robotics Laboratory (BRL). The robot's decision to release the object during the handover task is determined by an HMM [4]. Figure 1 presents the state diagram of the HMM corresponding to the basic handover interaction, where the label $L(s)$ is defined for each state.

Next, we initialise A, B and π of the HMM as follows. The process starts at state Robot not hold, so its initial distribution value π_1 is almost one, while the

other states have initial distribution values close to zero. The initial matrix A must encourage the transitions shown in Figure 1. To initialise B, we consider as observations the ordered pairs whose first and second components are the index and middle finger metacarpophalangeal joint motor current values, respectively. By the Cartesian product of these values, we obtain 56,404 observations. Since these observations are merged with the states to generate the DTMC passed to *Courcoubetis.hs*, and the size of a formula could grow considerably by associating the *next* operator with up to 56,404 observations, Marimba's execution is not practical under these circumstances. Vector quantisation [8] was used to reduce the number of observations to just 13, which were taken to initialise matrix B. Thus, the initial ordered pairs are grouped into 13 regions of the plane representing the observations.

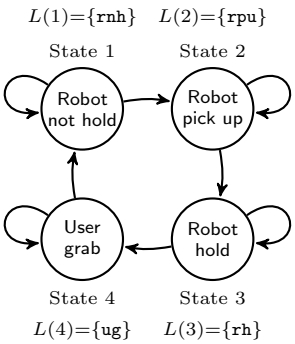

Fig. 1. The labelled states involved in the basic handover process.

To make reliable estimates, we collected observations from 50 handover experiments on Bert2. These observations were used to train the initial HMM with the reestimation method found in the solution of Rabiner's Problem 3 [9].

Liveness properties. A liveness property requires that a *good thing* happens during the execution of a system. For example, we would like to know whether *the model generates the sequence of observations* $\mathcal{O} = o_1, o_2, o_3, o_4$ *where* $o_1, o_2 \in \{3, 4, 6\}$ *and* $o_3, o_4 \in \{3, 4, 11\}$*, with probability greater than 0.88*, that is, $\mathcal{P}_{>0.88}(\mathbf{X}_{\{3,4,6\}}(\mathbf{X}_{\{3,4,6\}}(\mathbf{X}_{\{3,4,11\}}(\mathbf{X}_{\{3,4,11\}}\mathsf{true}))))$. Interestingly, this property is a generalisation of Rabiner's Problem 1 [9]. Marimba's execution for this property is found in Figure 2. The inputs are the trained HMM, defined in `ModelBert2.poctl`, and the previous formula. The output returned by Marimba is State 4. Hence, the model starting at state User grab is likely to generate $\mathcal{O}$.

```
Main> main
Enter the file name where the HMM is located.
examples/ModelBert2.poctl
Would you like to consider each state as if it were the initial
state, i.e., as if it had initial distribution value equal to 1? y/n: y
Enter the POCTL* formula we are interested in.
P[>0.88] (X_{3,4,6}(X_{3,4,6}(X_{3,4,11}(X_{3,4,11}T))))
The states that satisfy it are:
(Probability of satisfaction of each state:[4.998198505964186e-10,
            4.08659792160621e-6,7.508994137303159e-3,0.8915357419467848])
[4]
Do you want to continue checking more specifications? y/n: n
```

Fig. 2. Verifying a property with Marimba.

A second liveness property states that *with probability at least 0.9,* Bert2 *releases the object when the user grabs it.* The POCTL* formula for this property

is $\mathcal{P}_{\geq 0.9}(\mathtt{rh} \wedge (\mathtt{rh}\, \mathcal{U}\, (\mathtt{ug} \wedge \mathtt{ug}\, \mathcal{U}\, \mathtt{rnh})))$. Marimba outputs State 3, i.e., the specification is satisfied when the starting state is Robot hold. So, we expect Bert2 to hold the object, and let it go when the user grabs it.

Safety properties. A safety property establishes that a *bad thing* does not occur during the execution of a system. For instance, *with probability less than 0.05,* Bert2 *abandons its serving position with the user not grabbing the object*, that is, $\mathcal{P}_{<0.05}(\mathtt{rh} \wedge \mathbf{X}_{\Theta}(\mathtt{rnh} \vee \mathtt{rpu}))$, where Θ is the set of observations. Our model checker returns $\{1, 2, 3, 4\}$ as the set of states satisfying this property. We conclude that it is unlikely that the model, being at state Robot hold, reaches a state other than User grab, that is, Robot not hold or Robot pick up.

The satisfaction of the previous three specifications provides us with confidence that Bert2 reliably performs the handover interaction specified above.

On an Intel® Core™ i3 1.70GHz computer with 4GB in memory, Marimba takes 28.55s to compute the states satisfying the first liveness formula. The time required for checking the other two properties studied here is around 0.06s.

Further examples are given in the *examples* folder and *user's manual* that come with Marimba's source code.

4 Conclusions

Since the automatic verification of properties of HMMs seems to be an unattended problem, we present here Marimba, a Haskell implementation of the model checking algorithm for POCTL* [11]. This model checking algorithm was slightly modified to carry out its computations in a real program. Marimba's calculation is basically broken out in three stages that are coded in the modules *ModelChecker.hs*, *DirectApproach.hs* and *Courcoubetis.hs*, such that the involved components, steps and transformations are well arranged throughout the implementation. Finally, we have successfully applied Marimba to verify relevant properties of a handover interaction from the robot Bert2 to a human.

Acknowledgements. We gratefully acknowledge support from grants PAPIIT IN113013 and Conacyt 221341, and especially thank the BRL staff for their assistance operating the robot Bert2. E. Magid and K. Eder have been supported, in full and in part, respectively, by the UK EPSRC grant EP/K006320/1 ROBOSAFE: "Trustworthy Robotic Assistants".

References

1. C. Courcoubetis and M. Yannakakis, *The complexity of probabilistic verification*, J. ACM **42** (1995), no. 4, 857–907.
2. L. De Moura and N. Bjørner, *Z3: An efficient SMT solver*, Proceedings of the Theory and Practice of Software (TACAS '08), LNCS, Springer, 2008, pp. 337–340.
3. L. Erkok, *linearEqSolver: a library to solve systems of linear equations, using SMT solvers.*, `https://github.com/LeventErkok/linearEqSolver`.

4. E. C. Grigore, K. Eder, A. G. Pipe, C. Melhuish, and U. Leonards, *Joint action understanding improves robot-to-human object handover*, IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2013, pp. 4622–4629.
5. N. Hernández, *Model checking based on the hidden Markov model and its application to human-robot interaction*, Master's thesis, Universidad Nacional Autónoma de México, México, 2014, Available from `http://132.248.9.195/ptd2014/noviembre/303087692/Index.html`.
6. M. Kwiatkowska, G. Norman, and D. Parker, *PRISM 4.0: Verification of probabilistic real-time systems*, Proc. 23rd International Conference on Computer Aided Verification (CAV '11), LNCS, vol. 6806, Springer, 2011, pp. 585–591.
7. A. Lenz, S. Skachek, K. Hamann, J. Steinwender, A. G. Pipe, and C. Melhuish, *The BERT2 infrastructure: An integrated system for the study of human-robot interaction*, 10th IEEE-RAS International Conference on Humanoid Robots, IEEE, 2010, pp. 346–351.
8. Y. Linde, A. Buzo, and R. M. Gray, *An algorithm for vector quantizer design*, IEEE Transactions on Communications **28** (1980), 84–95.
9. L. R. Rabiner, *A tutorial on hidden Markov models and selected applications in speech recognition*, Proceedings of the IEEE **77** (1989), 257–286.
10. J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker, *Mathematical techniques for analyzing concurrent and probabilistic systems*, CRM Monograph Series, vol. 23, American Mathematical Society, 2004.
11. L. Zhang, H. Hermanns, and D. N. Jansen, *Logic and model checking for hidden Markov models*, Formal Techniques for Networked and Distributed Systems, FORTE 2005, LNCS, vol. 3731, Springer, 2005, pp. 98–112.

5 Appendix

Technical details and formal definitions concerning HMMs and the POCTL* formalism are presented next.

5.1 Hidden Markov Model

An HMM has two layers, one on top of the other. The stochastic process between states on the underlying layer is hidden, and can be seen only through the stochastic process on the external layer that effectively produces a visible sequence of observations.

Definition 1. *A labelled Hidden Markov Model* [9] *consists of a tuple* $\mathcal{H} = (S, A, \Theta, B, L, \pi)$*, where:*

- $S = \{S_0, S_1, \ldots, S_{n-1}\}$ *is a finite set of states;*
- A *is a state transition probability matrix, such that:*

$$A = \{a_{ij}\},\ a_{ij} \geq 0 \quad 0 \leq i, j \leq n-1, \qquad \sum\nolimits_{j=0}^{n-1} a_{ij} = 1 \quad 0 \leq i \leq n-1;$$

- $\Theta = \{v_0, v_1, \ldots, v_{m-1}\}$ *is a set of* m *observations;*
- B *is the observation probability matrix,* $B = \{b_j(k)\}$ *with*

$$b_j(k) = P[v_k \,|\, S_j], \quad 0 \leq j \leq n-1, \quad 0 \leq k \leq m-1;$$

- $L : S \to 2^{AP_{\mathcal{H}}}$ *maps states to sets of atomic propositions from a set* $AP_{\mathcal{H}}$*;*
- π *is an initial probability distribution over* S*, such that:*

$$\pi_i = P[q_0 = S_i] \geq 0 \qquad 0 \leq i \leq n-1, \qquad \sum\nolimits_{i=0}^{n-1} \pi_i = 1.$$

An execution of the system which is being modelled by an HMM is represented by a path.

Definition 2. *A path* [11] *is a sequence* $(s_0, o_0), (s_1, o_1), \ldots$*, where* $s_i \in S$*,* $o_i \in \Theta$*,* $a_{s_i s_{i+1}} > 0$ *and* $b_{s_i}(o_i) > 0$*,* $\forall i \geq 0$*. A path can be finite* (ω^{fin}) *or infinite* (ω)*.*

We denote the $(i+1)$st state of ω by $\omega_s(i)$, and the $(i+1)$st observation by $\omega_o(i)$. The suffix $(s_i, o_i), (s_{i+1}, o_{i+1}), \ldots$ of ω is denoted by $\omega[i]$. We denote the sets of all finite and infinite paths in $\mathcal{H}$, starting with a pair whose state is s, by $\mathsf{Path}_s^{\mathsf{fin},\mathcal{H}}$ and $\mathsf{Path}_s^{\mathcal{H}}$, respectively.

To quantify the probability that an HMM behaves in a certain way, we define the measure $\Pr_s$ over the set $\mathsf{Path}_s^{\mathcal{H}}$. The *basic cylinder set* induced by the cylinder $\omega^{\mathsf{fin}} = (s_0, o_0), (s_1, o_1), \ldots, (s_k, o_k)$ is defined as $C(\omega^{\mathsf{fin}}) = \{\omega \in \mathsf{Path}_s^{\mathcal{H}} \mid \forall i \in \{0, \ldots, k\}\ (\omega_s(i) = s_i \wedge \omega_o(i) = o_i)\}$. Let Σ_s be the smallest σ-algebra on $\mathsf{Path}_s^{\mathcal{H}}$ which contains all basic cylinder sets $C(\omega^{\mathsf{fin}})$, where $\omega^{\mathsf{fin}} = (s, o_0), \ldots, (s_k, o_k) \in \mathsf{Path}_s^{\mathsf{fin},\mathcal{H}}$. We define $\Pr_s$ on Σ_s as,

$$\Pr_s\Big(C\big((s, o_0), \ldots, (s_k, o_k)\big)\Big) = \pi_s b_s(o_0) \prod\nolimits_{i=1}^{k} a_{s_{i-1} s_i} b_{s_i}(o_i).$$

Let Σ be the smallest σ-algebra on $\mathsf{Path}^{\mathcal{H}}$ containing all basic cylinder sets, such that $\mathsf{Path}^{\mathcal{H}}$ is the set of paths in $\mathcal{H}$ with no constraint on the state of the initial pair. In [5], the probability measure $\Pr_{\mathcal{H}}$ on Σ is defined in terms of $\Pr_s$.

We quantify the probability that an HMM behaves in a certain way by identifying the set of paths that satisfy a formula, and then using $\Pr_{\mathcal{H}}$ (or $\Pr_s$).

5.2 POCTL*

The Probabilistic Observation Computational Tree Logic* (POCTL* [11]) has a *next* operator equipped with an observation constraint.

Definition 3 (Syntax). *Let $\mathcal{H} = (S, A, \Theta, B, L, \pi)$ be an HMM defined over the set of atomic propositions $AP_{\mathcal{H}}$. The syntax of POCTL* is defined as follows:*

$$\Phi ::= \mathsf{true} \mid \mathsf{false} \mid a \mid (\neg\Phi) \mid (\Phi \vee \Phi) \mid (\Phi \wedge \Phi) \mid (\mathcal{P}_{\bowtie p}(\phi)),$$
$$\phi ::= \Phi \mid (\neg\phi) \mid (\phi \vee \phi) \mid (\phi \wedge \phi) \mid (\mathbf{X}_{\mathsf{o}}\phi) \mid (\phi \mathcal{U}^{\leq n} \phi) \mid (\phi \mathcal{U} \phi),$$

where $a \in AP_{\mathcal{H}}$, $\mathsf{o} \in \Theta$, $n \in \mathbb{N}$, $p \in [0,1]$, and $\bowtie \in \{\leq, <, \geq, >\}$. We distinguish between state formulas Φ and path formulas ϕ.

Definition 4 (Semantics). *Let $\mathcal{H} = (S, A, \Theta, B, L, \pi)$ be an HMM. For any state $s \in S$, the satisfaction relation $\models$ is inductively defined as*

$$\begin{array}{ll} s \models \mathsf{true} & \forall s \in S, \\ s \not\models \mathsf{false} & \forall s \in S, \\ s \models a & \textit{iff } \ a \in L(s), \\ s \models \neg\Phi & \textit{iff } \ s \not\models \Phi, \end{array} \qquad \begin{array}{l} s \models \Phi_1 \vee \Phi_2 \ \ \textit{iff } \ s \models \Phi_1 \vee s \models \Phi_2, \\ s \models \Phi_1 \wedge \Phi_2 \ \ \textit{iff } \ s \models \Phi_1 \wedge s \models \Phi_2, \\ s \models \mathcal{P}_{\bowtie p}(\phi) \ \ \textit{iff } \ \Pr_s\{\omega \in \mathsf{Path}^{\mathcal{H}}_s \mid \omega \models \phi\} \bowtie p. \end{array}$$

For any path ω, the satisfaction relation is defined as

$$\begin{array}{ll} \omega \models \Phi & \textit{iff } \omega_s(0) \models \Phi, \\ \omega \models \neg\phi & \textit{iff } \omega \not\models \phi, \\ \omega \models \mathbf{X}_{\mathsf{o}}\phi & \textit{iff } \ \omega_o(0) = \mathsf{o} \wedge \omega[1] \models \phi, \\ \omega \models \phi_1 \mathcal{U}^{\leq n} \phi_2 & \textit{iff } \ \exists j \leq n.\ (\omega[j] \models \phi_2 \wedge \forall i < j.\ \omega[i] \models \phi_1), \\ \omega \models \phi_1 \mathcal{U} \phi_2 & \textit{iff } \ \exists j \geq 0.\ (\omega[j] \models \phi_2 \wedge \forall i < j.\ \omega[i] \models \phi_1). \end{array} \qquad \begin{array}{l} \omega \models \phi_1 \vee \phi_2 \ \textit{iff } \omega \models \phi_1 \vee \omega \models \phi_2, \\ \omega \models \phi_1 \wedge \phi_2 \ \textit{iff } \omega \models \phi_1 \wedge \omega \models \phi_2, \end{array}$$

Let $\Omega \subseteq \Theta$, we write $\mathbf{X}_{\Omega}\phi$ as a shorthand for $\bigvee_{\mathsf{o} \in \Omega} \mathbf{X}_{\mathsf{o}}\phi$. Therefore, $\omega \models \mathbf{X}_{\Omega}\phi$ iff $\omega_o(0) \in \Omega \wedge \omega[1] \models \phi$.

6 Model checking algorithm

Let $\mathcal{H} = (S, A, \Theta, B, L, \pi)$ be an HMM, s be a state in S, and Φ be a POCTL* state formula. Next, we explain a method to know whether $s \models \Phi$ holds or not.

6.1 Stage One

According to [11], the model checking algorithm starts by taking a most deeply nested state subformula Ψ of Φ, such that Ψ is not an atomic proposition. It is straightforward to find the states in S that satisfy Ψ when it is propositional. To obtain the states that satisfy Ψ when it is of the form $\mathcal{P}_{\bowtie p}(\phi)$, stage two of the model checker is invoked. Once we determine the states satisfying Ψ, their label is extended by a new atomic proposition a_Ψ. Next, Ψ is replaced by a_Ψ in Φ. The algorithm proceeds recursively, unless Φ itself is replaced by a_Φ; in such case the algorithm returns states s, with $a_\Phi \in L(s)$.

6.2 Stage Two

To identify the states that satisfy $\mathcal{P}_{\bowtie p}(\phi)$, we follow the direct approach that transforms the original $\mathcal{H}$ into a DTMC $\mathcal{D} = (S^{\mathcal{D}}, A^{\mathcal{D}}, L^{\mathcal{D}}, \pi^{\mathcal{D}})$, where

- $S^{\mathcal{D}} = S \times \Theta$,
- $A^{\mathcal{D}}((s,o),(s',o')) = a_{ss'} \cdot b_{s'}(o')$,
- $L^{\mathcal{D}}(s,o) = L(s) \cup \{\Omega \subseteq \Theta \mid o \in \Omega\}$,
- $\pi^{\mathcal{D}}_{(s,o)} = \pi_s \cdot b_s(o)$,

that is defined over the set of atomic propositions $AP_{\mathcal{D}} = AP_{\mathcal{H}} \cup \{\Omega \mid \Omega \subseteq \Theta\}$. The argument of $\mathcal{P}$, i.e., ϕ, is modified to obtain ϕ' in a way that every occurrence of $\mathbf{X}_\Omega \varphi$ is replaced by $\Omega \wedge \mathbf{X}\varphi$. Notice that Ω is a new atomic proposition defined in $AP_{\mathcal{D}}$.

6.3 Stage Three

As stated in [1,5], stage three recursively constructs a new DTMC $\mathcal{D}'$ by applying the transformations $C_{\mathbf{X}}$, $C_{\mathcal{U}}$ and $C_{\mathcal{U}^{\leq n}}$, which are performed for each occurrence of $\mathbf{X}$, $\mathcal{U}$ and $\mathcal{U}^{\leq n}$, respectively. To show how the transformations work, we focus here on $C_{\mathbf{X}}$. It takes $\mathbf{X}\varphi$ as an innermost subexpression of ϕ'. Then, it partitions the states of $\mathcal{D}$ into three disjoint subsets, $S^{\mathcal{D}} = S^{\mathsf{YES}} \cup S^{\mathsf{NO}} \cup S^{?}$, where:

- S^{YES} consists of the states whose transitions are only into states satisfying φ.
- S^{NO} consists of the states whose transitions are only into states satisfying $\neg\varphi$.
- $S^{?}$ consists of the states with transitions to both states satisfying φ and states satisfying $\neg\varphi$.

Let q_u denote the probability that $\mathbf{X}\varphi$ is satisfied starting from state $u \in S^{\mathcal{D}}$. We know that $q_u = 1$ if $u \in S^{\mathsf{YES}}$, and $q_u = 0$ if $u \in S^{\mathsf{NO}}$. Otherwise, $q_u = \sum_v A^{\mathcal{D}}(u,v)$, where the sum ranges over all successor states v of u satisfying formula φ. Let $\overline{q_u} = 1 - q_u$. Moreover, the new DTMC $\mathcal{D}'$ is defined over the set $AP_{\mathcal{D}'} = AP_{\mathcal{D}} \cup \{\xi\}$, where ξ is a new atomic proposition representing $\mathbf{X}\varphi$.

States of $\mathcal{D}'$. For each $u \in S^{\mathsf{YES}}$ there is a new state (u,ξ) in $\mathcal{D}'$. For each $u \in S^{\mathsf{NO}}$ there is a new state $(u,\neg\xi)$. And for each $u \in S^{?}$, there are two new states (u,ξ) and $(u,\neg\xi)$. We define $L^{\mathcal{D}'}(u,\xi) = L^{\mathcal{D}}(u) \cup \{\xi\}$ and $L^{\mathcal{D}'}(u,\neg\xi) = L^{\mathcal{D}}(u)$.

Transitions of $\mathcal{D}'$. The transition probability of $(u, \xi_1) \to (v, \xi_2)$, with $\xi_i \in \{\xi, \neg\xi\}$ and $i \in \{1, 2\}$, is defined as being equal to the probability that $\mathcal{D}$, being at state u, transitions next to state v, and starting from state v onward satisfies property ξ_2, conditioned on the event that in state u it satisfies property ξ_1.

Initial distribution of $\mathcal{D}'$. If $u \in S^{\mathsf{YES}} \cup S^{\mathsf{NO}}$, then $\pi^{\mathcal{D}'}_{(u,\xi_1)} = \pi^{\mathcal{D}}_u$. If $u \in S^?$, then there are two states in $\mathcal{D}'$ for u, namely (u, ξ) and $(u, \neg\xi)$, with initial probabilities $\pi^{\mathcal{D}}_u \cdot q_u$ and $\pi^{\mathcal{D}}_u \cdot \overline{q_u}$, respectively. Furthermore, ψ is obtained by replacing $\mathbf{X}\varphi$ by ξ in ϕ'.

If ϕ' originally has k temporal operators, the algorithm applies k times the appropriate transformations $C_{\mathbf{X}}$, $C_{\mathcal{U}}$ and $C_{\mathcal{U}^{\leq n}}$, to finally get the DTMC $\mathcal{D}^k$ and the propositional formula ψ^k. It is proved in [5] that $s \models \mathcal{P}_{\bowtie p}(\phi)$ iff

$$\sum_{o \in \Theta} \left(\sum_{\substack{\xi_{i_1} \in \{\xi_1, \neg\xi_1\} \\ \vdots \\ \xi_{i_k} \in \{\xi_k, \neg\xi_k\}}} \Pr\underbrace{((\ldots((s,o),\xi_{i_1}),\ldots),\xi_{i_k})}_{\sigma_0}\{\sigma \in \mathsf{Path}^{\mathcal{D}^k}_{\sigma_0} \mid \sigma \models \psi^k\} \right) \bowtie p.$$

Since ψ^k is a propositional formula, $\Pr_{\sigma_0}\{\sigma \in \mathsf{Path}^{\mathcal{D}^k}_{\sigma_0} \mid \sigma \models \psi^k\}$ is $\pi^{\mathcal{D}^k}_{\sigma_0}$.