Universidad Nacional Autónoma de México

Posgrado en Ciencia e Ingeniería de la Computación

Model checking based on the Hidden Markov Model and its application to human-robot interaction

T E S I S

Que para optar por el grado de:

Maestro en Ciencias (Computación)

P r e s e n t a:

Noé Salomón Hernández Sánchez

Tutor:
Dr. David Arturo Rosenblueth Laguette
Instituto de Investigaciones en Matemáticas Aplicadas y Sistemas

Co-Tutor:
Dra. Kerstin Inge Eder
Posgrado en Ciencia e Ingeniería de la Computación

México, D.F. Diciembre, 2014

Acknowledgements

First of all, I want to give many profound thanks to God for sharing with me the knowledge and understanding that made possible the development and conclusion of this thesis project.

My gratitude goes to my parents and siblings who motivate me to do my best in every aspect of my life. I realise their unconditional love and encouragement fuelled my desires to continue working on this thesis until its completion was reached.

I would like to deeply thank my supervisor, Dr. David Rosenblueth, for his valuable guidance and advice during this thesis. The meetings we held were always profitable and challenging. Also, our conversations will be remembered as interesting and instructive.

I want to give sincere thanks to my co-supervisor, Dr. Kerstin Eder. I regard with high value her feedback and suggestions on my work. Due to her support and collaboration, the time I spent at the University of Bristol had a relevant impact on me in academic and personal matters.

Many thanks to Dr. Evgeni Magid for his instruction and assistance handling BERT2 at the Bristol Robotics Laboratory (BRL).

I appreciate the help and advise offered by the BRL team and the RIVERAS group at University of Bristol. Special thanks to Dr. Dejanira Araiza, member of the RIVERAS group, for her contributions to this thesis.

I express my gratitude to the *Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica*, PAPIIT IN113013, for the support given to finish this work.

I am extremely thankful to my school, National Autonomous University of Mexico (UNAM). This institution provides me, and other tens of thousands of students, with an excellent education and marvellous opportunities to develop our capabilities to the fullest.

Thanks to Guillermina Flores Mateos for her kindness and sincere friendship.

Agradecimientos

Primero, quiero dar profundas gracias a Dios por compartir conmigo el conocimiento y la comprensión que hicieron posible el desarrollo y término de este proyecto de tesis.

Mi gratitud es hacia mis padres y hermanos quienes me motivan a dar lo mejor de mi en cada aspecto de la vida. Reconozco que su amor y ánimo incondicional me alentaron para continuar trabajando en esta tesis hasta alcanzar su término.

Me gustaría agradecer profundamente a mi asesor, Dr. David Rosenblueth, por su dirección y asesoramiento durante esta tesis. Las reuniones que tuvimos fueron siempre provechosas y estimulantes. Asimismo, nuestras pláticas serán recordadas como interesantes e instructivas.

Quiero ofrecer sinceras gracias a mi co-tutora, Dra. Kerstin Eder. Valoro mucho sus comentarios y sugerencias sobre mi trabajo. Debido a su apoyo y colaboración, el tiempo que pasé en la Universidad de Bristol tuvo un fuerte impacto en mi en cuestiones académicas y personales.

Muchas gracias al Dr. Evgeni Magid por su enseñanza y ayuda al manejar a Bert2 en el Laboratorio de Robótica de Bristol (BRL).

Aprecio mucho la ayuda y los consejos ofrecidos por el personal del BRL y por el grupo Riveras en la Universidad de Bristol. Gracias especiales a la Dra. Dejanira Araiza, miembro del grupo Riveras, por sus aportaciones a esta tesis.

Deseo expresar mi gratitud al *Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica*, PAPIIT IN113013, por el apoyo dado para finalizar este trabajo.

Estoy demasiado agradecido con mi escuela, la Universidad Nacional Autónoma de México (UNAM). Esta institución nos ofrece a mi, y a decenas de miles de otros estudiantes, una educación excelente y oportunidades únicas para desarrollar nuestras habilidades al máximo.

Gracias a Guillermina Flores Mateos por su amabilidad y sincera amistad.

Contents

Introduction

A *Model Checker* is a tool that verifies whether or not a property, written as a formula of a certain logic, is satisfied by a model of a system [22]. Several and diverse model checkers have been developed, amongst them we can mention:

- *NuSMV* [5], a SAT-based and BDD-based model checker, that expresses its specifications in LTL and CTL.
- *PRISM* [17], which manipulates various types of probabilistic models such as Discrete Time Markov Chains and Probabilistic Automata. Its specification language incorporates the temporal logics PCTL, CSL, LTL and PCTL*.
- *SPIN* [32], where the system models are described through the modelling language PROMELA (Process Meta Language), that emphasises the modelling of process synchronisation and coordination. Besides SPIN's built-in correctness requirements, other correctness properties can be specified as LTL formulas.
- *Kronos* [32], which checks whether a real-time system modelled by a Timed Automaton satisfies a timed property specified by a TCTL formula.

However, looking at the features of the previous model checkers, it seems that one of the most relevant probabilistic models has been ruled out, namely the Hidden Markov Model (HMM). Such a model was first studied in the late 1960s and early 1970s by Baum. Later on, Rabiner published a pivotal tutorial about HMM in the late 1980s [27]. As a consequence, a myriad of HMM's applications have been studied, such as speech recognition [19, 27], computational biology [15], compressed document processing [18], human gesture recognition [28], text recognition [8], and human-robot interaction [11, 12]. Would not it be interesting to study and analyse whether or not certain properties are satisfied in these situations? We think, the answer is an overwhelming 'yes'.

Naturally, it is highly interesting to consider how we can state and verify properties of HMM-based systems. This task, apparently forgotten by the previous model checkers, is investigated here. The main contribution of this project thesis is the construction, in the HASKELL programming language, of a model checker that makes use of HMMs to model the systems of interest, and whose specifications are formulated by the logic POCTL* [36]. Noticeably, as far as we know, there exists no model checker for this logic elsewhere, an assertion reinforced by first-hand comments of H. Hermanns, one of the creators of POCTL*. Additionally, to give a real-world example of the operation of our model checker, we address the basic human-robot handover interaction, where safety and liveness properties must be guaranteed to regard such activity as reliable to be carried out in real life [10, 11, 12]. We study the handover interaction based on the robot BERT2 [20].

The research questions considered in this thesis are:

- How is the model checking algorithm for POCTL* modified, with respect to its original description, when we seek to implementing it?
- What are the conveniences and difficulties, in terms of code generation, of implementing the model checker in HASKELL?
- Can we get an HMM, suitable for our model checker, that captures the basic handover interaction on BERT2?
- How likely is the object dropped when the basic handover task is performed on BERT2?

In this thesis we have two aims. First, we implement a model checker that verifies properties of systems represented by HMMs. Such properties are stated through the expressive power of POCTL*. Secondly, using the resulting tool of the first aim, we verify whether or not safety and liveness properties hold under the scenario in which a robot hands an object over to a human.

The first aim comprises the implementation of the model checking algorithm for the logic POCTL*. This algorithm is outlined in three main stages and follows the idea given in Zhang's thesis [35]. The input of the model checker consists of both an HMM, which takes the form of a file with extension `.poctl`, and a POCTL* formula. We accomplish our second aim by building, out of a series of experiments run on BERT2, an HMM that models the basic handover process. Then, we formulate safety and liveness properties for this human-robot interaction. Finally, the model checker is provided with the previous HMM and properties. The evaluation of its corresponding outcomes should allow us to conclude that this handover process is trustworthy.

The rest of this thesis is organised as follows. In Chapter 1, HMMs are defined and their three fundamental problems are shown with their particular solutions. Here, probability measures on HMMs, needed by the POCTL* semantics, are also studied. In Chapter 2, the logic POCTL* is given, together with its syntactic rules and semantic relations; moreover, some important sublogics and examples of the POCTL* expressiveness are also available. The model checking algorithm is described in Chapter 3. Besides the mere algorithm, this chapter gives the reasoning on which every single instruction of the model checker is built upon. In Chapter 4, we focus on the actual HASKELL implementation of the model checker. We take advantage of this functional programming language to ease the coding task. Lastly, the explanation of the handover process, run on the BERT2's platform, is contained in Chapter 5. It provides details on the construction of an HMM representing this interaction. Also, safety and liveness properties related to the handover situation are drawn on. Both the HMM and the properties are passed to our model checker, whose results are evaluated in terms of the handover task.

Chapter 1

Hidden Markov Model

The impediment of knowing all the elements that affect a system or agent is an uncertainty that must be studied; moreover, it is interesting to consider how the environment changes with time. Among the more relevant models used to represent this probabilistic situation is the Markov Model. In this chapter, we will introduce the Hidden Markov Model. We start briefly discussing the Discrete Time Markov Chain (DTMC), then we move on to present the Hidden Markov Model.

1.1 Discrete Time Markov Model

Suppose that we have a system with n states $(S_0, S_1, \ldots, S_{n-1})$. With each of them we associate an observable event happening in our environment in a way that, for each discrete time interval, a transition between states is produced. We name the instance in which the transition takes place as $t = 0, 1, \ldots$, and the state in which the system is at the moment t as q_t. Furthermore, let the system be a first-order Markov process, i.e., the current state q_t depends only on the previous state q_{t-1} and not on states before to q_{t-1}, so the next equality holds:

$$P[q_t = S_j \mid q_{t-1} = S_i, q_{t-2} = S_k, \ldots q_0 = S_l] = P[q_t = S_j \mid q_{t-1} = S_i].$$

We also assume that the DTMC is homogeneous, in other words, the above equation is true for every t. Hence we can define a DTMC giving the transition probabilities between states $a_{ij} = P[q_t = S_j \mid q_{t-1} = S_i]$.

To study additional properties of DTMCs we label the states with atomic propositions that represent interesting assertions of the system.

Definition 1.1. A *labelled Discrete Time Markov Chain* is a tuple $\mathcal{D} = (S, A, L, \pi)$ 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:

$$\begin{aligned} A &= \{a_{ij}\} && 0 \leq i,j \leq n-1, \\ a_{ij} &\geq 0 && 0 \leq i,j \leq n-1, \\ \sum_{j=0}^{n-1} a_{ij} &= 1 && 0 \leq i \leq n-1; \end{aligned}$$

- $L : S \to 2^{AP}$, L is a labelling function that maps a state in S to a subset of the set of atomic propositions AP;

- π is an initial probability distribution over S, such that:

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

∎

Consider the following example. Say that we are interested in knowing what the weather in Bristol, UK, is going to be like using DTMCs. To simplify our problem let us say that we can choose from three different states: rainy (S_0), drizzly (S_1) and cloudy (S_2); and that the weather conditions in Bristol are checked once a day.

The transition probability matrix A is

$$A = \begin{pmatrix} 0.4 & 0.4 & 0.2 \\ 0.3 & 0.4 & 0.3 \\ 0.2 & 0.5 & 0.3 \end{pmatrix}.$$

The initial distribution is $\pi = [0.35, 0.4, 0.25]$.

Finally, the set of atomic propositions AP contains two elements: p, which declares that people in Bristol go to the pub; and q, which indicates that people in Bristol stay home watching a film. So, the labelling function is

$$L(S_0) = \{q\}, \quad L(S_1) = \{p\}, \quad L(S_2) = \{p\}.$$

We would like to know the probability of the weather forecast for the next four days to be *cloudy – cloudy – rainy – drizzly.* Let $\mathcal{O}$ be the observation sequence $\mathcal{O} = \{S_2, S_2, S_0, S_1\}$. The desired probability is found using the product rule (see Appendix A.1) three times, so

$$\begin{aligned} P(\mathcal{O} \,|\, \mathcal{D}) &= P[S_2, S_2, S_0, S_1 \,|\, \mathcal{D}] \\ &= P[S_2] \cdot P[S_2 \,|\, S_2] \cdot P[S_0 \,|\, S_2] \cdot P[S_1 \,|\, S_0] \\ &= \pi_2 \cdot a_{22} \cdot a_{20} \cdot a_{01} \\ &= 0.25 \cdot 0.3 \cdot 0.2 \cdot 0.4 \\ &= 0.006. \end{aligned}$$

Now, we want to know the probability that, in a three-day period, on the third day people stay at home watching a film. We know that this only happens if, on the third day, the atomic proposition q holds. In other words, if the last state of the sequence $\mathcal{O} = \{q_0, q_1, q_2\}$ is S_0. Applying marginalisation (see Appendix A.1) and the product rule, we obtain

$$\begin{aligned}
P[q_2 = S_0 \,|\, \mathcal{D}] &= \sum_{i,j \in \{0,1,2\}} P[q_0 = S_i, q_1 = S_j, q_2 = S_0 \,|\, \mathcal{D}] \\
&= \sum_{i,j \in \{0,1,2\}} P[S_i] \cdot P[S_j \,|\, S_i] \cdot P[S_0 \,|\, S_j] \\
&= \sum_{i,j \in \{0,1,2\}} \pi_i \cdot a_{ij} \cdot a_{j0} \\
&= \pi_0 \cdot a_{00} \cdot a_{00} + \pi_0 \cdot a_{01} \cdot a_{10} + \pi_0 \cdot a_{02} \cdot a_{20} + \\
&\quad\ \pi_1 \cdot a_{10} \cdot a_{00} + \pi_1 \cdot a_{11} \cdot a_{10} + \pi_1 \cdot a_{12} \cdot a_{20} + \\
&\quad\ \pi_2 \cdot a_{20} \cdot a_{00} + \pi_2 \cdot a_{21} \cdot a_{10} + \pi_2 \cdot a_{22} \cdot a_{20} \\
&= 0.112 + 0.12 + 0.0725 \\
&= 0.3045.
\end{aligned}$$

1.2 Hidden Markov Model

Let us recall that the states in a DTMC represent an observable event. This restriction prevents us from modelling many interesting processes. Thus, we extend this model so that the observation currently seen is the output of a probabilistic function over the states. So a Hidden Markov Model (HMM) is a doubly embedded stochastic[1] process [27, 31, 35, 36]. We can say that an HMM has two layers, one on top of the other. The stochastic process (a DTMC) on the underlying layer is not visible, i.e., it is hidden, and can be seen only through the stochastic process on the external layer that effectively produces a visible sequence of observations.

The Hidden Markov Model has important and many applications such as speech recognition, DNA sequence analysis and robot control. Before giving a formal definition, we present next an example of the use of HMMs.

Suppose that we are interested in predicting the food offered at our favourite restaurant. The choice depends on the chef's mood. Say that he or she can only be either happy (H) or upset (U). Furthermore his or her mood changes stochastically. Nonetheless, the chef is busy all the time and we do not want to unexpectedly enter the kitchen just to ask for the chef's mood. Moreover, we know that the main course the chef cooks depends probabilistically on his or her mood. The main course the chef prepares is any of the following: steak (S), pasta (P) or haddock fillet (F). We wonder: what

[1] A process is stochastic if it is not deterministic, where the uncertainty of the possible outcomes is expressed in terms of probabilities [29].

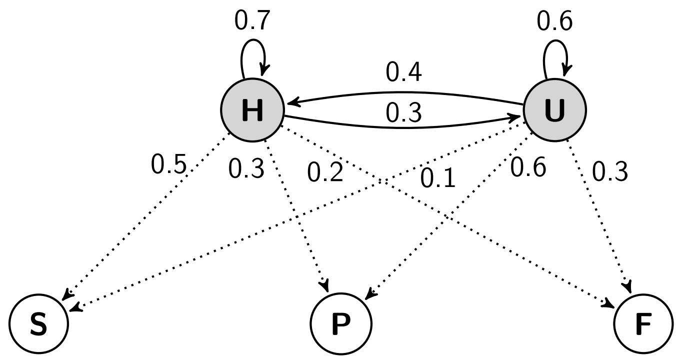


Figure 1.1: An HMM diagram for the chef's mood problem.

is the probability that the main course for the next five days is P, P, S, F and S, respectively?

It is clear that the states of the HMM modelling this situation are H and U. We consider the observations as the possible meals S, P and F. The chef's change of mood from one day to the other is described by the transition probability matrix:

$$A = \begin{array}{c c} & \begin{array}{cc} H & U \end{array} \\ \begin{array}{c} H \\ U \end{array} & \begin{pmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{pmatrix} \end{array}.$$

The relation between the chef's mood and the meal he or she cooks is pictured in the observation probability matrix:

$$B = \begin{array}{c c} & \begin{array}{ccc} S & P & F \end{array} \\ \begin{array}{c} H \\ U \end{array} & \begin{pmatrix} 0.5 & 0.3 & 0.2 \\ 0.1 & 0.6 & 0.3 \end{pmatrix} \end{array}.$$

In Figure 1.1 the transition probabilities between states are expressed with a continuous line, while the observation probabilities are depicted with dotted lines.

Finally, assume that on the first day someone tells us how likely the chef's mood is going to be on that day, thus we get the following initial state distribution

$$\pi = \begin{array}{c} \begin{array}{cc} H & U \end{array} \\ \begin{pmatrix} 0.6 & 0.4 \end{pmatrix} \end{array}.$$

Definition 1.2. The *labelled Hidden Markov Model* consists of a tuple $\mathcal{H} = (S, A, \Theta, B, L, \pi)$, where:

- (S, A, L, π) is a labelled Discrete Time Markov Chain (see Definition 1.1);
- Θ is a set of m observations, such that $\Theta = \{v_0, v_1, \ldots, v_{m-1}\}$;
- B is the observation probability matrix of dimension $n \times m$, such that B is a collection of observation symbol distributions in the states of the model, i.e.,

$B = \{b_j(k)\}$ with,

$$b_j(k) = P[v_k \text{ at the instant } t \,|\, q_t = S_j], \quad \begin{array}{l} 0 \le j \le n-1, \\ 0 \le k \le m-1. \end{array}$$

Additionally, the probabilities $b_j(k)$ are independent of t.

■

The procedure an HMM follows to generate the observation sequence $\mathcal{O} = \{o_0, o_1, \ldots, o_{T-1}\}$, with each $o_i \in \Theta$, is described below:

1. For $t = 0$, an initial state is chosen, $q_t = S_i$, according to the initial distribution π.

2. We pick an observation, $o_t = v_k$, using the distribution $b_i(k)$; notice that this observation distribution is in the state S_i.

3. The model transits to the state $q_{t+1} = S_j$, as dictated by the state transition probability a_{ij}.

4. If $t < T - 1$, we update the time $t = t + 1$ and jump to step 2; otherwise the procedure finishes.

1.2.1 The three basic problems for HMMs

In 1989, Rabiner wrote an outstanding tutorial on HMMs, where he shows the three fundamental problems originally asked by Jack Ferguson. Furthermore, Rabiner gives elegant and efficient solutions to these problems in his tutorial [27], which we also present after stating the three respective problems.

Let $\mathcal{O} = \{o_0, o_1, \ldots, o_{T-1}\}$ be an observation sequence and $\mathcal{H}$ be an HMM, the three basic problems are:

Problem 1. How do we efficiently compute the probability of $\mathcal{O}$ given the model $\mathcal{H}$, i.e., $P[\mathcal{O} \,|\, \mathcal{H}]$?

Problem 2. How do we choose an optimal state sequence that best *explains* the observations $\mathcal{O}$? That is, how do we unveil the hidden state sequence?

Problem 3. How do we adjust the model parameters such that $P[\mathcal{O} \,|\, \mathcal{H}]$ is maximised?

Solution to Problem 1

To solve Problem 1, we need to find $P[\mathcal{O} \,|\, \mathcal{H}]$, i.e., the probability that the observation sequence $\mathcal{O} = \{o_0, o_1, \ldots, o_{T-1}\}$ is produced, given the model $\mathcal{H}$. Observe that the desired probability does not choose a particular state sequence, hence we must look at all possible state sequences of length T. Let $Q = q_0 q_1 \ldots q_{T-1}$ be one of such sequences.

If we assume that Q has occurred, the probability is computed by taking the values of the current state producing the respective observation,

$$P[\mathcal{O} \,|\, Q, \mathcal{H}] = \prod_{t=0}^{T-1} P[o_t \,|\, q_t, \mathcal{H}] = b_{q_0}(o_0) b_{q_1}(o_1) \cdots b_{q_{T-1}}(o_{T-1}). \tag{1.1}$$

Notice that Q starts in q_0, then it moves onto q_1, after that to q_2, and so on until it reaches q_{T-1}. Therefore, the probability of Q being generated is

$$P[Q \,|\, \mathcal{H}] = \pi_{q_0} a_{q_0 q_1} a_{q_1 q_2} \cdots a_{q_{T-2} q_{T-1}}. \tag{1.2}$$

Using the formula for conditional probability (see Appendix A.1) we get the equation,

$$P[\mathcal{O}, Q \,|\, \mathcal{H}] = P[\mathcal{O} \wedge Q \,|\, \mathcal{H}] = \frac{P[\mathcal{O} \wedge Q \wedge \mathcal{H}]}{P[\mathcal{H}]}.$$

Also, it is true that,

$$P[\mathcal{O} \,|\, Q, \mathcal{H}] \; P[Q \,|\, \mathcal{H}] = \frac{P[\mathcal{O} \wedge Q \wedge \mathcal{H}]}{\cancel{P[Q \wedge \mathcal{H}]}} \cdot \frac{\cancel{P[Q \wedge \mathcal{H}]}}{P[\mathcal{H}]} = \frac{P[\mathcal{O} \wedge Q \wedge \mathcal{H}]}{P[\mathcal{H}]}.$$

Consequently,

$$P[\mathcal{O}, Q \,|\, \mathcal{H}] = P[\mathcal{O} \,|\, Q, \mathcal{H}] \; P[Q \,|\, \mathcal{H}].$$

Observe that we already know the values of $P[\mathcal{O} \,|\, Q, \mathcal{H}]$ and $P[Q \,|\, \mathcal{H}]$ by Equations (1.1) and (1.2), respectively.

To calculate the probability of $\mathcal{O}$, given the model $\mathcal{H}$, we add up the joint probability over Q on the left-hand side of the previous equation. This task is known as marginalisation (see [29] and Appendix A.1). Accordingly,

$$\begin{aligned} P[\mathcal{O} \,|\, \mathcal{H}] &= \sum_{\text{for all } Q} P[\mathcal{O}, Q \,|\, \mathcal{H}] \qquad (1.3) \\ &= \sum_{\text{for all } Q} P[\mathcal{O} \,|\, Q, \mathcal{H}] \; P[Q \,|\, \mathcal{H}] \\ &= \sum_{\text{for all } Q} \pi_{q_0} b_{q_0}(o_0) a_{q_0 q_1} b_{q_1}(o_1) \cdots a_{q_{T-2} q_{T-1}} b_{q_{T-1}}(o_{T-1}). \end{aligned}$$

This direct computation has a poor performance, since it involves on the order of $2Tn^T$ calculations, because we have n^T possible state sequences and, according to the above summation, for each such state sequence about $2T$ computations are executed. The resulting running time is computationally unfeasible.

There exists, however, an efficient method to compute this probability. The process is known as the *forward* algorithm, for which we define the forward variable $\alpha_t(i)$ as the probability of the partial observation sequence $o_0 o_1 \ldots o_t$ and state S_i at time t, given the model $\mathcal{H}$, i.e.,

$$\alpha_t(i) = P[o_0, o_1, \ldots, o_t, q_t = S_i \,|\, \mathcal{H}].$$

The definition of the forward variable is inductive, as shown next:

Initialisation: We need to start in state S_i and see the observation o_0 there. That is,

$$\alpha_0(i) = \pi_i b_i(o_0), \qquad 0 \le i \le n-1.$$

Induction: We wish to find the value of $\alpha_t(j)$, the probability of the partial observation $o_0 \dots o_t$ and state S_j at instance t. Now let us consider the probability that in the previous instance the state S_i is reached and the observation sequence $o_0 \dots o_{t-1}$ is seen, i.e., $\alpha_{t-1}(i)$, then we move from S_i to S_j, i.e., a_{ij}. Once in state S_j, we should observe o_t, i.e., $b_j(o_t)$. Consequently,

$$\alpha_t(j) = \left[\sum_{i=0}^{n-1} \alpha_{t-1}(i) a_{ij}\right] b_j(o_t), \quad 1 \le t \le T-1, \qquad 0 \le j \le n-1.$$

Termination: Finally, applying marginalisation to $P[\mathcal{O} \,|\, \mathcal{H}]$ we obtain,

$$\begin{aligned} P[\mathcal{O} \,|\, \mathcal{H}] &= \sum_{i=0}^{n-1} P[\mathcal{O}, q_{T-1} = S_i \,|\, \mathcal{H}] \\ &= \sum_{i=0}^{n-1} \alpha_{T-1}(i). \end{aligned}$$

To find out the complexity of the forward algorithm, we note that the forward variable, $\alpha_t(j)$, needs $n+1$ multiplications at the induction step. This is done for every time $1, 2, \dots, T-1$. Moreover, this is carried out for the n states of the system, hence we have $n(n+1)(T-1)$ multiplications. We must also consider the multiplication required by the initialization step of α for all n states. Thus, the total number of multiplications used by the forward variable is $n(n+1)(T-1)+n$. A similar analysis shows that the number of additions used is $n(n-1)(T-1)$. So, the computation required by this method is on the order of n^2T, a much smaller number of operations than the ones involved in the naive approach.

By the forward algorithm, we can now address the question we were wondering when discussing the chef's mood problem. The probability that the main course for the next five days is P, P, S, F and S, respectively, i.e., $P[\mathcal{O} \,|\, \mathcal{H}]$, where $\mathcal{O} = \{P, P, S, F, S\}$ and $\mathcal{H}$ is the model given as an example in Section 1.2, is $P[\mathcal{O} \,|\, \mathcal{H}] = \sum_{i=0}^{1} \alpha_4(i) = 0.00431$.

Solution to Problem 2

There are several possible ways of solving Problem 2, that is, finding the "optimal" state sequence $Q = \{q_0 q_1 \cdots q_{T-1}\}$ associated with the given observation sequence $\mathcal{O} = \{o_0 \cdots o_{T-1}\}$, because there are several possible optimality criteria. The one discussed here focuses on finding the *single* best state sequence, i.e., it maximises $P[Q \,|\, \mathcal{O}, \mathcal{H}]$. By the product rule (see Appendix A.1), $P[Q \,|\, \mathcal{O}, \mathcal{H}] = P[Q, \mathcal{O} \,|\, \mathcal{H}] \,/\, P[\mathcal{O} \,|\, \mathcal{H}]$. Thus this optimality criterion is equivalent to maximising $P[Q, \mathcal{O} \,|\, \mathcal{H}]$. The process that computes this state sequence is known as the *Viterbi algorithm* and is described next.

We define the quantity

$$\delta_t(i) = \max_{q_0, \cdots, q_{t-1}} P[q_0 \cdots q_t = S_i, o_0 \cdots o_t \,|\, \mathcal{H}],$$

that is, $\delta_t(i)$ is the highest probability along a single path, at time t, which accounts for the first t observations and ends in state S_i. By induction we have

$$\delta_t(j) = \max_{0 \leq i \leq n-1} [\delta_{t-1}(i) a_{ij}] \cdot b_j(o_t).$$

We realise the need to keep track of the argument i which maximised the last expression, for each t and j. That is why we use the array ψ, where the cell $\psi_t(j)$ holds the state index that maximises the probability of the observation sequence $o_0 o_1 \cdots o_{t-1}$ and state S_j at time t.

Another important aspect to highlight is the fact that underflow is likely to happen since we are going to multiply numbers close to zero. We cope with this problem by simply taking logarithms and adding them up.

Below, the complete Viterbi algorithm is stated; next to its conventional definition, its logarithmic version appears.

Initialisation: For $0 \leq i \leq n-1$,

$$\text{Conventional: } \delta_0(i) = \pi_i b_i(o_0), \qquad \text{Logarithmic: } \delta_0(i) = \log(\pi_i) + \log(b_i(o_0)).$$

Induction: For $1 \leq t \leq T-1$ and $0 \leq j \leq n-1$,

$$\text{Conventional: } \begin{cases} \delta_t(j) &= \max\limits_{0 \leq i \leq n-1} [\delta_{t-1}(i) a_{ij}] \, b_j(o_t), \\ \psi_t(j) &= \arg\max\limits_{0 \leq i \leq n-1} [\delta_{t-1}(i) a_{ij}]. \end{cases}$$

$$\text{Logarithmic: } \begin{cases} \delta_t(j) &= \max\limits_{0 \leq i \leq n-1} [\delta_{t-1}(i) + \log(a_{ij})] \; + \log(b_j(o_t)), \\ \psi_t(j) &= \arg\max\limits_{0 \leq i \leq n-1} [\delta_{t-1}(i) + \log(a_{ij})]. \end{cases}$$

Termination:

$$\text{Both versions: } \begin{cases} P^* &= \max\limits_{0 \leq i \leq n-1} [\delta_{T-1}(i)], \\ q^*_{T-1} &= \arg\max\limits_{0 \leq i \leq n-1} [\delta_{T-1}(i)]. \end{cases}$$

Path backtracking: For $t = T-2, T-3, \ldots, 0$,

$$\text{Both versions: } q^*_t = \psi_{t+1}(q^*_{t+1}).$$

We should notice that this algorithm is an instance of *dynamic programming* and that it resembles the forward algorithm. The major difference is the maximisation calculation over previous states performed in the Viterbi algorithm, whereas in the definition of the forward variable a summation is computed instead.

Solution to Problem 3

To deal with Problem 3 we use a procedure called *backward variable*, $\beta_t(i)$, that is reminiscent of the forward variable, $\alpha_t(i)$, shown in the solution of Problem 1. The backward variable is defined as

$$\beta_t(i) = P[o_{t+1}o_{t+2}\ldots o_{T-1} \,|\, q_t = S_i, \mathcal{H}].$$

It is interpreted as the probability of the partial observation sequence form $t+1$ to the end, given state S_i at time t and the model $\mathcal{H}$. We give an inductive definition of $\beta_t(i)$ as follows:

Initialisation: We define for $0 \leq i \leq n-1$,

$$\beta_{T-1}(i) = 1.$$

Induction: To have been in state S_i at time t, and the observation sequence from time $t+1$ on to occur, we have to examine all possible states S_j at time $t+1$, the cost of going from state S_i to S_j, as well as that of producing the observation o_{t+1} in state S_j. Finally, we consider the remaining partial observation sequence beginning in state S_j, i.e., $\beta_{t+1}(j)$. So,

$$\beta_t(i) = \sum_{j=0}^{n-1} a_{ij}b_j(o_{t+1})\beta_{t+1}(j), \qquad t = T-2, T-3, \ldots, 0, \quad 0 \leq i \leq n-1.$$

The running time required to compute $\beta_t(i)$, for $0 \leq t \leq T-1$ and $0 \leq i \leq n-1$, is on the order of n^2T calculations.

Baum-Welch algorithm

Now we address the problem of adjusting the model matrices A, B and π, to maximise the probability of the observation sequence $\mathcal{O}$ given the original model $\mathcal{H}$. We will depict an iterative procedure known as the Baum-Welch method that produces a new model $\overline{\mathcal{H}} = (S, \overline{A}, \Theta, \overline{B}, L, \overline{\pi})$ such that $P[\mathcal{O} \,|\, \overline{\mathcal{H}}]$ is locally maximised. To that end we must define two more probability values, namely $\xi_t(i,j)$ and $\gamma_t(i)$.

We first define $\xi_t(i,j)$ as the probability of being in state S_i at time t, and in state S_j at time $t+1$, given the observation sequence $\mathcal{O}$ and the model $\mathcal{H}$, that is,

$$\xi_t(i,j) = P[q_t = S_i, q_{t+1} = S_j \,|\, \mathcal{O}, \mathcal{H}].$$

Thanks to the conditional probability formula, we see that the previous equation can be written as,

$$\xi_t(i,j) = \frac{P[q_t = S_i, q_{t+1} = S_j, \mathcal{O} \,|\, \mathcal{H}]}{P[\mathcal{O} \,|\, \mathcal{H}]}.$$

It is clear that $\xi_t(i,j)$ can be easily expressed using the forward and backward variables as shown next:

$$\begin{aligned}\xi_t(i,j) &= \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{P[\mathcal{O}\,|\,\mathcal{H}]}\\ &= \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\displaystyle\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}.\end{aligned}$$

We also define $\gamma_t(i)$ as the probability of being in state S_i at time t, given the observation sequence $\mathcal{O}$ and the model $\mathcal{H}$, that is, $\gamma_t(i) = P[q_t = S_i\,|\,\mathcal{O},\mathcal{H}]$. Applying the conditional probability formula to the right-hand side results in

$$\gamma_t(i) = \frac{P[q_t = S_i, \mathcal{O}\,|\,\mathcal{H}]}{P[\mathcal{O}\,|\,\mathcal{H}]}.$$

This equation can be simply expressed in terms of the forward and backward variables, i.e.,

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P[\mathcal{O}\,|\,\mathcal{H}]} = \frac{\alpha_t(i)\beta_t(i)}{\displaystyle\sum_{i=0}^{n-1}\alpha_t(i)\beta_t(i)}.$$

Additionally, we can relate $\gamma_t(i)$ to $\xi_t(i,j)$ by summing over the states S_j, giving $\gamma_t(i) = \sum_{j=0}^{n-1}\xi_t(i,j)$.

A further examination of $\gamma_t(i)$ reveals that if we sum it over the time t, ignoring the time slot $t = T-1$, we get a quantity which can be interpreted as the expected number of transitions made from state S_i. Thus,

$$\sum_{t=0}^{T-2}\gamma_t(i) = \text{expected number of transitions from } S_i.$$

Likewise, summation of $\xi_t(i,j)$ over t, with $0 \le t \le T-2$, can be interpreted as the expected number of transitions from state S_i to state S_j. That is,

$$\sum_{t=0}^{T-2}\xi_t(i,j) = \text{expected number of transitions from } S_i \text{ to } S_j.$$

A method for reestimation of the parameters of an HMM can be drawn via the two newly defined variables and the idea of counting event occurrences. As a result we have:

$$\overline{\pi}_i = \text{expected number of times in state } S_i \text{ at the initial intance} = \gamma_0(i),$$

$$\overline{a}_{ij} = \frac{\text{expected number of transitions from state } S_i \text{ to state } S_j}{\text{expected number of transitions from state } S_i} = \frac{\sum_{t=0}^{T-2} \xi_t(i,j)}{\sum_{t=0}^{T-2} \gamma_t(i)},$$

$$\overline{b}_j(k) = \frac{\text{expected number of times in } S_j \text{ and observing symbol } v_k}{\text{expected number of times in state } S_j} = \frac{\sum_{\substack{t=0 \\ \text{s.t. } o_t = v_k}}^{T-1} \gamma_t(j)}{\sum_{t=0}^{T-1} \gamma_t(j)}.$$

The reestimated model $\overline{\mathcal{H}}$ is more likely to have produced the observation sequence $\mathcal{O}$ than model $\mathcal{H}$, i.e., $P(\mathcal{O} \,|\, \overline{\mathcal{H}}) \geq P(\mathcal{O} \,|\, \mathcal{H})$. From that we see that if we iterate this process using $\overline{\mathcal{H}}$ in place of $\mathcal{H}$ and carry out the reestimation operations once again, we can then improve the probability of $\mathcal{O}$ being generated from the model until some threshold is reached.

Since $\alpha_t(i)$ consists of a sum of several terms involving the multiplication of $\alpha_{t-1}(j)$, a_{ji} and $b_i(o_t)$, which are less than 1 by far, each $\alpha_t(i)$ starts to head exponentially to zero as t gets bigger. For sufficiently large values of t, the computation of $\alpha_t(i)$ will exceed the precision range of practically any computer. Therefore a scaling procedure must be implemented. In order to keep the scaled $\alpha_t(i)$ within the precision range handled by the computer, the scaling procedure multiplies $\alpha_t(i)$ by a scaling coefficient c_t independent of i. For the backward variable $\beta_t(i)$ a similar scaling process has to be done. This causes the scaling coefficients to cancel out at the end of the reestimation process.

The scaled value of $\alpha_t(i)$ is kept in the variable $\widehat{\alpha}_t(i)$. For $0 \leq t \leq T-1$ we have,

$$c_t = \frac{1}{\sum_{i=0}^{n-1} \overset{\star}{\alpha}_t(i)}, \qquad \widehat{\alpha}_t(i) = c_t \overset{\star}{\alpha}_t(i), \quad 0 \leq i \leq n-1,$$

where $\overset{\star}{\alpha}_0(i) = \pi_i b_i(o_0)$ and $\overset{\star}{\alpha}_t(i) = \left[\sum_{j=0}^{n-1} \widehat{\alpha}_{t-1}(j) a_{ji}\right] b_i(o_t)$, if $1 \leq t \leq T-1$.

The scaled βs are found according to the equation $\widehat{\beta}_t(i) = c_t \overset{\star}{\beta}_t(i)$, where $\overset{\star}{\beta}_{T-1}(i) = 1$, and $\overset{\star}{\beta}_t(i) = \sum_{j=0}^{n-1} a_{ij} b_j(o_{t+1}) \widehat{\beta}_{t+1}(j)$ if $t = T-2, \ldots, 0$. Observe that the same scaling factor c_i is used on both $\widehat{\alpha}_t(i)$ and $\widehat{\beta}_t(i)$.

Moreover, the previously studied reestimation equations incorporate the scaled variables as:

$$\overline{\pi}_i = \gamma_0(i) = \sum_{j=0}^{n-1} \xi_0(i,j) = \frac{\sum\limits_{j=0}^{n-1} \widehat{\alpha}_0(i) a_{ij} b_j(o_1) \widehat{\beta}_1(j)}{\sum\limits_{i=0}^{n-1}\sum\limits_{j=0}^{n-1} \widehat{\alpha}_0(i) a_{ij} b_j(o_1) \widehat{\beta}_1(j)},$$

$$\overline{a}_{ij} = \frac{\sum\limits_{t=0}^{T-2} \xi_t(i,j)}{\sum\limits_{t=0}^{T-2} \gamma_t(i)} = \frac{\sum\limits_{t=0}^{T-2} \xi_t(i,j)}{\sum\limits_{t=0}^{T-2}\sum\limits_{j=0}^{n-1} \xi_t(i,j)} = \frac{\sum\limits_{t=0}^{T-2} \widehat{\alpha}_t(i) a_{ij} b_j(o_{t+1}) \widehat{\beta}_{t+1}(j)}{\sum\limits_{t=0}^{T-2}\sum\limits_{j=0}^{n-1} \widehat{\alpha}_t(i) a_{ij} b_j(o_{t+1}) \widehat{\beta}_{t+1}(j)},$$

$$\overline{b}_j(k) = \frac{\sum\limits_{\substack{t=0 \\ \text{s.t. } o_t = v_k}}^{T-1} \gamma_t(j)}{\sum\limits_{t=0}^{T-1} \gamma_t(j)} = \frac{\sum\limits_{\substack{t=0 \\ \text{s.t. } o_t = v_k}}^{T-1} \frac{\widehat{\alpha}_t(j)\widehat{\beta}_t(j)}{c_t}}{\sum\limits_{t=0}^{T-1} \frac{\widehat{\alpha}_t(j)\widehat{\beta}_t(j)}{c_t}}.$$

From the previous equations just the one related to $\overline{a}_{ij}$ is presented in Rabiner's tutorial. The equations for $\overline{\pi}_i$ and $\overline{b}_j(k)$ are simply found using the new scaled $\widehat{\alpha}_t(j)$ and $\widehat{\beta}_t(j)$. In addition, we will show that in the product $\widehat{\alpha}_t(j)\widehat{\beta}_t(j)$ the term c_t is multiplied twice, that is why we divide by c_t when computing $\overline{b}_j(k)$. First, we see that $\widehat{\alpha}_t(i) = \prod_{k=0}^{t} c_k \alpha_t(i)$ and $\widehat{\beta}_t(i) = \prod_{k=t}^{T-1} c_k \beta_t(i)$ can be proven by induction on t. Consequently,

$$\overline{b}_i(k) = \frac{\sum\limits_{\substack{t=0 \\ \text{s.t. } o_t = v_k}}^{T-1} \frac{\widehat{\alpha}_t(j)\widehat{\beta}_t(j)}{c_t}}{\sum\limits_{t=0}^{T-1} \frac{\widehat{\alpha}_t(j)\widehat{\beta}_t(j)}{c_t}} = \frac{\sum\limits_{\substack{t=0 \\ \text{s.t. } o_t = v_k}}^{T-1} \frac{\prod_{k=0}^{t} c_k \alpha_t(i) \cdot \prod_{k=t}^{T-1} c_k \beta_t(i)}{c_t}}{\sum\limits_{t=0}^{T-1} \frac{\prod_{k=0}^{t} c_k \alpha_t(i) \cdot \prod_{k=t}^{T-1} c_k \beta_t(i)}{c_t}}$$

$$= \frac{\sum\limits_{\substack{t=0 \\ \text{s.t. } o_t = v_k}}^{T-1} \frac{c_t \prod_{k=0}^{T-1} c_k \, \alpha_t(i)\beta_t(i)}{c_t}}{\sum\limits_{t=0}^{T-1} \frac{c_t \prod_{k=0}^{T-1} c_k \, \alpha_t(i)\beta_t(i)}{c_t}} = \frac{\sum\limits_{\substack{t=0 \\ \text{s.t. } o_t = v_k}}^{T-1} \alpha_t(i)\beta_t(i)}{\sum\limits_{t=0}^{T-1} \alpha_t(i)\beta_t(i)}.$$

The last formula is precisely the original reestimation expression for $\overline{b}_i(k)$, so the scaling process works well. Using a similar argument we can prove that $\overline{\pi}_i$ and $\overline{a}_{ij}$ also compute the reestimated values correctly.

In Chapter 5, we will train an HMM using a kind of model that goes from one state to another in a sequential manner, for which a single observation sequence is not enough to adequately train the model. The reason is that, for any state within the model, only a small number of observations is allowed to be brought on; from there, a transition may be made to the next state and more observations may be yielded. Therefore, in order to have sufficient data to make reliable estimates of the model parameters, multiple observation sequences have to be considered. The modification of the reestimation-scaling formulas is straightforward. We simply append to the formulas we already have an extra summation that combines the individual frequencies of occurrence for each observation sequence, as is explained below. The details on how to train an HMM with multiple observation sequences are found in [34].

Let $\mathbf{O}$ be a set of observation sequences such that

$$\mathbf{O} = \{\mathcal{O}^{(0)}, \mathcal{O}^{(1)}, \ldots, \mathcal{O}^{(K-1)}\},$$

where $\mathcal{O}^{(k)} = \{o_0^{(k)}, o_1^{(k)}, \ldots, o_{T-1}^{(k)}\}$ are individual observation sequences. Assume further that each observation sequence $\mathcal{O}^{(k)}$ is independent from the others.

Finally, taking into account the multiple observation sequences, the final reestimation-scaling formulas, used in Chapter 5 are

$$\overline{\pi}_i = \frac{1}{K}\sum_{k=0}^{K-1}\left(\frac{\displaystyle\sum_{j=0}^{n-1}\widehat{\alpha}_0^{(k)}(i)a_{ij}b_j(o_1^{(k)})\widehat{\beta}_1^{(k)}(j)}{\displaystyle\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\widehat{\alpha}_0^{(k)}(i)a_{ij}b_j(o_1^{(k)})\widehat{\beta}_1^{(k)}(j)}\right),$$

$$\overline{a}_{ij} = \frac{\displaystyle\sum_{k=0}^{K-1}\sum_{t=0}^{T-2}\widehat{\alpha}_t^{(k)}(i)a_{ij}b_j(o_{t+1}^{(k)})\widehat{\beta}_{t+1}^{(k)}(j)}{\displaystyle\sum_{k=0}^{K-1}\sum_{t=0}^{T-2}\sum_{j=0}^{n-1}\widehat{\alpha}_t^{(k)}(i)a_{ij}b_j(o_{t+1}^{(k)})\widehat{\beta}_{t+1}^{(k)}(j)},$$

$$\overline{b}_j(m) = \frac{\displaystyle\sum_{k=0}^{K-1}\sum_{\substack{t=0\\ \text{s.t. } o_t^{(k)}=v_m}}^{T-1}\frac{\widehat{\alpha}_t^{(k)}(j)\widehat{\beta}_t^{(k)}(j)}{c_t}}{\displaystyle\sum_{k=0}^{K-1}\sum_{t=0}^{T-1}\frac{\widehat{\alpha}_t^{(k)}(j)\widehat{\beta}_t^{(k)}(j)}{c_t}}.$$

K-means

The K-means procedure is another technique for adjusting parameters. It requires an initial model $\mathcal{H}$ and a set of training observation sequences. These observations sequences are segmented into states according to the optimal state sequence calculated with the Viterbi algorithm and the model $\mathcal{H}$. This segmentation into states allows us to count the number of times that each state S_i was visited and the number of times that such state saw the observation v_m generated. Hence, reestimation formulas can be found from this counting of event occurrences. The new value for $\overline{\pi}_i$ is obtained by dividing the number of times the optimal state sequence started at S_i by the total number of training observation sequences. We assign to $\overline{a}_{ij}$ the result of dividing the number of transitions from S_i to S_j by the total number of transitions from S_i to any state. Finally, we compute $\overline{b}_j(v_m)$ as the number of observations with index m, i.e., v_m, generated in state S_j divided by the total number of observations seen in state S_j.

The K-means procedure is clear and simple to program; furthermore, it essentially generates identical estimates values for the HMMs as its Baum-Welch counterpart. Besides these features, there are some advantages of the K-means technique over the Baum-Welch approach. For instance, it is about an order of magnitude faster and scaling is by far easier, because the forward and backward variables are only used in the context of the Viterbi algorithm that applies logarithms to avoid underflow.

We seek to increase the reliability of the reestimated model computed in Chapter 5. So we implemented both the Baum-Welch and K-means methods given here, expecting that their corresponding outputs would be particularly alike.

1.2.2 Belief states

According to Zhang's thesis [35], to have a concise representation of what we know about the current state, we summarise the history of seen observations in a *belief state*, which is a probability distribution over S. In other words, a *belief state* is a way to describe what we know about the state, given the history of observations. The set of all distributions over S is the set of all possible belief states, it is denoted by $\mathcal{B}$. We define next the belief state at time T, i.e., $\mathfrak{b}_T$.

Definition 1.3. Let $\mathcal{O} = \{o_0, \ldots, o_T\}$ be a sequence of observations. The belief state $\mathfrak{b}_T$ at time T is the distribution over S at time T given the history of observations $\mathcal{O}$, i.e.,

$$\mathfrak{b}_T(j) = P[q_T = S_j \mid \mathcal{O}, \mathcal{H}] \qquad 0 \leq j \leq n-1.$$

∎

To compute the value of a belief state at time $t = 0, \ldots, T$ we see that, in the first instance, the history of observations has just one element o_0 and that the state is determined by the initial distribution π. For the belief state at time $t = 1, \ldots, T$, we have to consider the history of observations seen so far. Hence we can give an inductive definition for $\mathfrak{b}_t$ that depends on the previous belief state $\mathfrak{b}_{t-1}$ and the current observation o_t. In his thesis, Zhang shows that

Initialisation: The belief state at time 0 is,

$$\mathfrak{b}_0(j) = \frac{\pi_j b_j(o_0)}{K_0},$$

with K_0 a normalisation constant, such that,

$$\begin{aligned}
K_0 &= P[o_0 \mid \mathcal{H}] && \text{(marginalization)}\\
&= \textstyle\sum_{j=0}^{n-1} P[o_0, q_0 = S_j \mid \mathcal{H}] && \text{(product rule)}\\
&= \textstyle\sum_{j=0}^{n-1} P[q_0 = S_j \mid \mathcal{H}] \cdot P[o_0 \mid q_0 = S_j, \mathcal{H}]\\
&= \textstyle\sum_{j=0}^{n-1} \pi_j b_j(o_0).
\end{aligned}$$

Induction: The new belief state $\mathfrak{b}_t$ is defined on terms of the previous belief state $\mathfrak{b}_{t-1}$ as,

$$\mathfrak{b}_t(j) = \frac{\sum_{i=0}^{n-1} \mathfrak{b}_{t-1}(i) a_{ij} b_j(o_t)}{K_t} \qquad 1 \le t \le T,$$

where K_t is a normalisation constant, such that,

$$\begin{aligned}
&K_t =\\
&= P[o_t \mid o_0, \ldots, o_{t-1}, \mathcal{H}] && \text{(marg.)}\\
&= \textstyle\sum_{j=0}^{n-1} P[o_t, q_t = S_j \mid o_0, \ldots, o_{t-1}, \mathcal{H}] && \text{(product rule)}\\
&= \textstyle\sum_{j=0}^{n-1} P[q_t = S_j \mid o_0, \ldots, o_{t-1}, \mathcal{H}] \cdot P[o_t \mid q_t = S_j, o_0, \ldots, o_{t-1}, \mathcal{H}] && \text{(marg.)}\\
&= \textstyle\sum_{j=0}^{n-1} \Big(\sum_{i=0}^{n-1} P[q_t = S_j, q_{t-1} = S_i \mid o_0, \ldots, o_{t-1}, \mathcal{H}] \cdot b_j(o_t)\Big) && \text{(product rule)}\\
&= \textstyle\sum_{j=0}^{n-1} \Big(\sum_{i=0}^{n-1} P[q_{t-1} = S_i \mid o_0, \ldots, o_{t-1}, \mathcal{H}] \cdot\\
&\qquad \cdot P[q_t = S_j \mid q_{t-1} = S_i, o_0, \ldots, o_{t-1}, \mathcal{H}] \cdot b_j(o_t)\Big)\\
&= \textstyle\sum_{j=0}^{n-1} \Big(\sum_{i=0}^{n-1} \mathfrak{b}_{t-1}(i) a_{ij} b_j(o_t)\Big).
\end{aligned}$$

Interestingly, the initialisation and induction definitions of the belief state at time t, $\mathfrak{b}_t(j)$, and of the forward variable, $\alpha_t(j)$, are extremely similar.

1.2.3 Paths and probability measures

The treatment we present here for paths and probability measures is taken from [6, 16, 35].

Definition 1.4. An execution of the system being modelled by an HMM is represented by a path. Formally, a path is a sequence of ordered pairs $(s_0, o_0), (s_1, o_1), \ldots$, where $s_i \in S$, $o_i \in \Theta$, $a_{s_i s_{i+1}} > 0$ and $b_{s_i}(o_i) > 0$, for all $i \ge 0$. A path can be either finite (ω^{fin}) or infinite (ω), we will deal with infinite paths unless stated otherwise. ∎

We denote the $(i+1)$st state of the path ω as $\omega_s(i)$; likewise, to indicate the $(i+1)$st observation of ω we write $\omega_o(i)$. The suffix of ω that ignores the first i pairs, i.e., the suffix that captures the sequence $(s_i, o_i), (s_{i+1}, o_{i+1}), \ldots$, is denoted by $\omega[i]$. Moreover, we denote the sets of all finite and infinite paths in $\mathcal{H}$ starting with a pair whose first component is state s as $\mathsf{Path}_s^{fin,\mathcal{H}}$ and $\mathsf{Path}_s^{\mathcal{H}}$, respectively. When $\mathcal{H}$ is obvious from the context, we remove it from the notation.

To study the probabilistic behaviour of the HMM, we need to be able to determine the probability that certain paths are taken. To this aim we define the measure $\Pr_s$ over the set Path_s. A finite path determines a basic event, known as cylinder [16]. We define the *basic cylinder set* induced by the cylinder $\omega^{fin} = (s_0, o_0), (s_1, o_1), \ldots, (s_k, o_k)$ as,

$$C(\omega^{fin}) \overset{def}{=} \{\omega \in \mathsf{Path}_s \mid \forall i \in \{0, \ldots, k\} \; (\omega_s(i) = s_i \wedge \omega_o(i) = o_i)\}.$$

Let Σ_s be the smallest σ-algebra on Path_s (see Appendix A.1) which contains all basic cylinder sets $C(\omega^{fin})$, where ω^{fin} can be any path in Path_s^{fin}. We define the measure $\Pr_s$ on Σ_s as,

$$\begin{aligned}\Pr_s\big(C(\omega^{fin})\big) &= \Pr_s\Big(C\big((s, o_0), \ldots, (s_k, o_k)\big)\Big)\\ &= \pi_s b_s(o_0) \prod_{i=1}^{k} a_{s_{i-1}s_i} b_{s_i}(o_i).\end{aligned}$$

From equation (1.3), we notice that the measure $\Pr_s$ is the probability of the observation sequence $\mathcal{O} = o_0, \ldots, o_k$ and state sequence $Q = s, \ldots, s_k$, given the model $\mathcal{H}$, i.e., $P[\mathcal{O}, Q \mid \mathcal{H}]$.

Let Σ be the smallest σ-algebra on Path, i.e., the set of paths in $\mathcal{H}$ such that the first component of the initial pair can be any state. We define the probability measure $\Pr_{\mathcal{H}}$ over Σ in terms of the measures $\Pr_s$ next. Since $\bigcup_{s \in S} C\big((s, o_0), \ldots, (s_k, o_k)\big) \in \Sigma$ and the cylinder sets in this family are disjoint, we have

$$\begin{aligned}&\Pr_{\mathcal{H}}\Big(\bigcup_{s\in S} C\big((s, o_0), \ldots, (s_k, o_k)\big)\Big)\\ =&\sum_{s\in S} \Pr_{\mathcal{H}}\Big(C\big((s, o_0), \ldots, (s_k, o_k)\big)\Big) \quad \text{(By Definition A.2)}\\ =&\sum_{s\in S} \Pr_s\Big(C\big((s, o_0), \ldots, (s_k, o_k)\big)\Big).\end{aligned}$$

We can now quantify the probability that an HMM behaves in a specified fashion by identifying the set of paths which satisfy the corresponding specification and using the associated measure $\Pr_{\mathcal{H}}$ (or $\Pr_s$).

Additionally, we extend the measure $\Pr_s$ with respect to belief states as follows. Let $\mathfrak{b}$ be a belief state, i.e., $\mathfrak{b} \in \mathcal{B}$, ω^{fin} be the cylinder $(s_0, o_0), \ldots, (s_k, o_k)$ and $C(\omega^{fin})$ be a basic cylinder set induced by ω^{fin}, then

$$\Pr_{\mathfrak{b}}\big(C(\omega^{fin})\big) = \sum_{s\in S} \mathfrak{b}(s) \cdot \Pr_s\big(C(\omega^{fin})\big).$$

Observe that the only moment we use a belief state is in Chapter 2. In fact it is a trivial belief state that assigns the value of 1 to a certain state and 0 to the others.

Interestingly, the measure with respect to the belief state, $\mathrm{Pr}_{\mathfrak{b}}$, was originally purposed by Zhang *et al.* in [35, 36] to weight the values Pr_s by $\mathfrak{b}$. Furthermore, in [35, 36] the calculation of Pr_s does not consider the initial distribution π, whereas we indeed account for it when computing Pr_s, as stated above, that is, we weight the values Pr_s by π. Therefore, to get the same effect that Zhang intended, we can integrate the belief state into the probability measure $\mathrm{Pr}_{\mathcal{H}}$ by taking $\mathfrak{b}$ instead of π.

In this chapter we studied the Hidden Markov Model, which can be an appropriate representation of diverse phenomena. We also saw how to find the probability of an observation sequence using the forward variable, we devised a manner to unveil the optimal sequence of states given a sequence of observations via the Viterbi algorithm, and two methods of training the HMM were covered. Moreover, we discussed belief states and developed tools to measure the probability that an HMM behaves in a certain way, when some paths are taken. In the next chapter we will study a formal language suitable to specify system properties for HMMs, namely the logic POCTL*.

Chapter 2

The Logic POCTL*

We are about to introduce a formal language that will allow us to specify properties of an HMM. Actually, we wish to specify properties involving the states of the model, i.e., our desire is to express formulas over the underlying DTMC, but we are also interested in studying properties of the external stochastic process, where the observations are perceived. This language is the Probabilistic Observation Computational Tree Logic* (POCTL* [35, 36]), an extension of PCTL*, which in turn is a probabilistic version of the logic CTL* [3]. POCTL* has the peculiarity that the *next* operator is equipped with an observation constraint. This is the way in which we add observations to our specifications, thus the formula $\mathbf{X}_o\phi$ means that the next observation is o (we can also interpret it as the next transition being labelled with an action o) and the remaining path satisfies ϕ.

Typically, the definition of a language is divided in two parts: *syntax*, where the structure and correct formation of the elements of the language is established using some grammatical rules; and *semantics*, that explains, based on the syntax, the meaning and expected behaviour of such elements, and the relation among themselves. Hence, we present first the syntax of POCTL* and later on its semantics.

2.1 Syntax

Definition 2.1. Let $\mathcal{H} = (S, A, \Theta, B, L, \pi)$ be an HMM and AP a set of atomic propositions. The syntax of POCTL* is defined as follows:

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

where $a \in AP$, $o \in \Theta$, $n \in \mathbb{N}$, $p \in [0, 1]$ and $\bowtie \in \{\leq, <, \geq, >\}$. ∎

To get rid of the parentheses without introducing ambiguity, we assume that the unary connectives (consisting of $\neg$, $\mathbf{X}$ and $\mathcal{P}$) bind most tightly. Next in order comes $\mathcal{U}^{\leq n}$; after that $\mathcal{U}$; then comes $\wedge$; and finally $\vee$. However, we will keep the inner parentheses of the probabilistic operator, $\mathcal{P}_{\bowtie p}(\phi)$, as well as the parentheses that are used to override the operator precedence just defined.

We distinguish several categories of formulas. Φ is for state formulas, ϕ for path formulas, and ϵ for belief state formulas. Since we always know the current belief state in HMMs, although the current state is uncertain, we want to know if some probabilistic properties are valid in belief states. Let $\epsilon = \mathcal{P}_{\bowtie p}(\phi)$ be a belief state formula and $\mathfrak{b}$ be a belief state. Intuitively, $\mathfrak{b}$ satisfies ϵ if the probability measure with respect to $\mathfrak{b}$ of the set of paths satisfying ϕ is within the threshold imposed by $\bowtie p$.

The grammar presented here is slightly different from the one used in [35] and [36]. In these papers the algorithm of the model checker for POCTL* is designed, whereas this thesis focuses on implementing such algorithm, hence it is likely we will sometimes choose a distinct approach that suits our needs and limitations arising while programming. For instance, we add syntactic rules for true, false and the disjunction operator, features missed in the original grammar. Furthermore, we explicitly have a constructor for the unbounded until operator, which Zhang and his colleagues declare it is obtained by taking n equals to ∞ in $\phi_1 \mathcal{U}^{\leq n} \phi_2$, clearly this idea is infeasible in practice.

2.2 Semantics

For an HMM $\mathcal{H}$, a state s and POCTL* state formula Φ, we write $s \models_{\mathcal{H}} \Phi$ to indicate that s satisfies Φ assuming $\mathcal{H}$ is the model being used. We also can say that Φ is true in s over $\mathcal{H}$. Likewise, for a path ω satisfying a path formula ϕ, we write $\omega \models_{\mathcal{H}} \phi$, and for a belief state formula ϵ that holds in a belief state $\mathfrak{b}$, we declare it as $\mathfrak{b} \models_{\mathcal{H}} \epsilon$. When the considered HMM $\mathcal{H}$ is obvious from the context, we leave the model out from the expression and write $s \models \Phi, \omega \models \phi$ and $\mathfrak{b} \models \epsilon$ instead.

Definition 2.2. 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 over the state formulas as

$$\begin{array}{ll}
s \models \mathsf{true} & \forall s \in S \\
s \not\models \mathsf{false} & \forall s \in S \\
s \models a & \text{iff } a \in L(s) \\
s \models \neg\Phi & \text{iff } s \not\models \Phi \\
s \models \Phi_1 \vee \Phi_2 & \text{iff } s \models \Phi_1 \vee s \models \Phi_2 \\
s \models \Phi_1 \wedge \Phi_2 & \text{iff } s \models \Phi_1 \wedge s \models \Phi_2 \\
s \models \epsilon & \text{iff } \mathfrak{b}_s \models \epsilon,
\end{array}$$

where $\mathfrak{b}_s$ is a belief state such that $\mathfrak{b}_s(s) = 1$ and $\mathfrak{b}_s(s') = 0$, with $s' \neq s$. Now, for any belief state $\mathfrak{b}$, we define the satisfaction relation as

$$\begin{array}{ll}
\mathfrak{b} \models \mathcal{P}_{\bowtie p}(\phi) & \text{iff } \Pr_{\mathfrak{b}}\{\omega \in \mathsf{Path} \mid \omega \models \phi\} \bowtie p \\
\mathfrak{b} \models \neg\epsilon & \text{iff } \mathfrak{b} \not\models \epsilon \\
\mathfrak{b} \models \epsilon_1 \vee \epsilon_2 & \text{iff } \mathfrak{b} \models \epsilon_1 \vee \mathfrak{b} \models \epsilon_2 \\
\mathfrak{b} \models \epsilon_1 \wedge \epsilon_2 & \text{iff } \mathfrak{b} \models \epsilon_1 \wedge \mathfrak{b} \models \epsilon_2.
\end{array}$$

Observe that $\Pr_{\mathfrak{b}}\{\omega \in \mathsf{Path} \mid \omega \models \phi\}$ denotes the probability measure of the set of all paths which satisfy ϕ, whose initial state is weighted according to $\mathfrak{b}$. Finally, for any path ω, the satisfaction relation is defined as

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

▮

Let Ω be a set of observations, i.e., $\Omega \subseteq \Theta$, we write $\mathbf{X}_\Omega\phi$ as a shorthand for $\vee_{o\in\Omega}\mathbf{X}_o\phi$, given the previous semantics we have $\omega \models \mathbf{X}_\Omega\phi$ iff $\omega_o(0) \in \Omega \wedge \omega[1] \models \phi$.

We also allow path formulas to include the $\Diamond\phi$ operator and its bounded variant $\Diamond^{\leq n}\phi$. Intuitively, $\Diamond\phi$ means that ϕ is eventually satisfied, whereas $\Diamond^{\leq n}\phi$ means that ϕ is satisfied within n units of time. They are syntax sugar and can be expressed in terms of the POCTL* grammar given in Definition 2.1 as follows:

$$\Diamond\phi \equiv \mathsf{true}\,\mathcal{U}\phi, \qquad \Diamond^{\leq n}\phi \equiv \mathsf{true}\,\mathcal{U}^{\leq k}\phi.$$

Moreover, we add the temporal logic operator $\Box\phi$ and its bounded variant $\Box^{\leq n}\phi$. The idea is that for $\Box\phi$ the path formula ϕ holds always along every step of the path, whereas the $\Box^{\leq n}\phi$ says that ϕ is true in the first n steps of the path. Using the fact that negations are allowed in path formulas, we can express the operator $\Box$ as follows:

$$\Box\phi \equiv \neg\Diamond\neg\phi, \qquad \Box^{\leq n}\phi \equiv \neg\Diamond^{\leq n}\neg\phi.$$

2.3 The sublogics

The exhaustive list of sublogics of POCTL* is presented in [35]. Here we focus on two of them, namely POCTL and the Quantitative OLTL Specification language, or QOS for short. As we will see, they are going to play an important role when studying the model checker for POCTL*.

2.3.1 POCTL

The logic POCTL is a constrained version of POCTL*, in which every presence of the temporal operators ($\mathbf{X}$, $\mathcal{U}^{\leq n}$ and $\mathcal{U}$) is immediately attached to the probabilistic operator $\mathcal{P}$. As a result, the syntax of POCTL is

$$\begin{array}{rcl} \Phi & := & \mathsf{true} \mid \mathsf{false} \mid a \mid \neg\Phi \mid \Phi \vee \Phi \mid \Phi \wedge \Phi \mid \epsilon \\ \phi & := & \mathbf{X}_o\Phi \mid \Phi\mathcal{U}^{\leq n}\Phi \mid \Phi\mathcal{U}\Phi \\ \epsilon & := & \mathcal{P}_{\bowtie p}(\phi) \mid \neg\epsilon \mid \epsilon \vee \epsilon \mid \epsilon \wedge \epsilon, \end{array}$$

where $a \in AP$, $o \in \Theta$, $n \in \mathbb{N}$, $p \in [0,1]$ and $\bowtie \in \{\leq, <, \geq, >\}$.

Using our notation for $\mathbf{X}_\Omega\Phi$, with $\Omega \subseteq \Theta$, we see that $\mathbf{X}\Phi \equiv \mathbf{X}_\Theta\Phi$. This observation-free variant of the next operator is the one used in the logic PCTL (see Appendix C.1), thus PCTL is in turn a sublogic of POCTL. A relevant remark arises when considering the formula $\mathcal{P}_{<0.5}(\mathbf{X}_o(a\mathcal{U}^{\leq 27}b))$. It is a valid POCTL* formula, but a false assertion for POCTL.

2.3.2 QOS

To define the logic QOS we must first study the logic it comes from, that is, OLTL.

OLTL

In Observation LTL (OLTL), we can have any combination of temporal operators and propositional formulas; nonetheless, formulas involving the probability operator are not allowed. Formally, OLTL formulas are expressed through the following grammar

$$\phi := \mathsf{true} \mid \mathsf{false} \mid a \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \mathbf{X}_o\phi \mid \phi\mathcal{U}^{\leq n}\phi \mid \phi\mathcal{U}\phi.$$

Note that OLTL may be seen as an extension of the logic LTL (see Appendix C.2), where the next operator, $\mathbf{X}_o$, has been equipped with an observation constraint.

We are now able to define the the language QOS. We generate QOS formulas as pairs of the form $(\phi, \bowtie p)$, where ϕ is an OLTL formula, $\bowtie \in \{\leq, <, >, \geq\}$ and $p \in [0,1]$. Note that the logic QOS has been defined as a quantitative version of OLTL. Similarly, the logic QLS (see the end of Appendix C.2) is a quantitative version of LTL.

Let $\mathcal{H} = (S, A, \Theta, B, L, \pi)$ be an HMM and s be a state in S. The semantics of QOS formulas is stated next

$$\mathcal{H}, s \models (\phi, \bowtie p) \Longleftrightarrow \Pr_s\{\omega \in \mathsf{Path}_s \mid \omega \models \phi\} \bowtie p.$$

2.4 The expressive power of POCTL*

We show next a list of properties that let us appreciate how we may write specifications through the logic POCTL*.

- With a probability of at least 0.85, the model produces the sequence of observations $\{o_0, o_1, \ldots, o_{T-1}\}$.

  The corresponding POCTL* formula, with nested parenthesis omitted, is

  $$\mathcal{P}_{\geq 0.85}(\mathbf{X}_{o_0}\mathbf{X}_{o_1}\ldots\mathbf{X}_{o_{T-1}}\mathsf{true}).$$

  There are three remarks about this formula. First, it describes a bounded version of Rabiner's Problem 1 related to HMMs (see Section 1.2.1). Second, the formula above can be seen as either a state formula or belief state formula. Then, to compute the appropriate value, the probability measure is chosen accordingly. And third, any sublogic of POCTL* discussed in Section 2.3 is unable to express this property.

- The probability that the state sequence $\{q_0, q_1, \ldots, q_{T-1}\}$ produces the observation sequence $\{o_0, o_1, \ldots, o_{T-1}\}$ is at most 0.25.

  To find the corresponding POCTL* formula, we associate with each state q_i the atomic proposition a_{q_i}, which is true only in q_i. Observe though that not all states, q_i, are necessarily different. Therefore, this property results in the formula

  $$\mathcal{P}_{\leq 0.25}\Big(a_{q_0} \wedge \mathbf{X}_{o_0}\Big(a_{q_1} \wedge \mathbf{X}_{o_1}\big(\ldots(a_{q_{T-1}} \wedge \mathbf{X}_{o_{T-1}}(\mathsf{true}))\ldots\big)\Big)\Big).$$

  Note that this formula attempts to unveil the hidden state sequence, situation addressed by Rabiner's Problem 2.

In this chapter, the POCTL* syntax and semantics were given, together with two of its sublogics: POCTL and QOS, that we will use later in this thesis. To demonstrate how POCTL* can be used to specify relevant properties of an HMM, we revealed, in the last section, some examples of formulas belonging to this logic. The next chapter describes in detail the model checking algorithm for POCTL*; in it thorough explanations on how and why the algorithm works are provided.

Chapter 3

Model Checking for POCTL*

Model checking is the process of answering the question of whether a state s satisfies a formula Φ with $\mathcal{H}$ as the model, i.e., $s \models_{\mathcal{H}} \Phi$, or simply $s \models \Phi$. Up to this point, we are acquainted with all the elements of that expression, that is, $\mathcal{H}$ is an HMM that captures the essentials of the system being studied, s is a state of this model, Φ is a specification written as a POCTL* state formula and $\models$ is the underlying satisfaction relation. What we are still missing is the *verification method* to establish whether $\mathcal{H}$ in s satisfies Φ. This chapter is dedicated to present such method, known as model checker, for the logic POCTL*.

The model checking algorithm, roughly speaking, can be divided in three stages. The first one follows the same lines given in [35]. We deviate from [35] for the second and third stages, that deal with QOS formulas, and consider the procedure depicted in [6, 36]. We also provide proofs of critical results that support the decisions made along the algorithm.

It is important to note that we agreed to specify properties of an HMM using state formulas, since we are interested in specifying properties related to the states of the model.

Remark 3.1. Looking at the original syntax of POCTL* given in Definition 2.1, we notice that the belief state formulas ϵ can be generated by Φ adding the probabilistic operator $\mathcal{P}$ to the state formulas rule. Thus we simplify the grammar in the following manner

$$\begin{array}{lcl} \Phi & := & \mathsf{true} \mid \mathsf{false} \mid a \mid \neg\Phi \mid \Phi \vee \Phi \mid \Phi \wedge \Phi \mid \mathcal{P}_{\bowtie p}(\phi) \\ \phi & := & \Phi \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \mathbf{X}_o\phi \mid \phi\mathcal{U}^{\leq n}\phi \mid \phi\mathcal{U}\phi. \end{array}$$

The semantic relation for states is extended to handle the probabilistic operator as

$$\begin{array}{lll} s \models \mathcal{P}_{\bowtie p}(\phi) & \text{iff} & \mathfrak{b}_s \models \mathcal{P}_{\bowtie p}(\phi) \\ & \text{iff} & \Pr_{\mathfrak{b}_s}\{\omega \in \mathsf{Path} \mid \omega \models \phi\} \bowtie p \\ & \text{iff} & \sum_{s_i \in S} \mathfrak{b}_s(s_i) \cdot \Pr_{s_i}\{\omega \in \mathsf{Path}_{s_i} \mid \omega \models \phi\} \bowtie p. \end{array}$$

As stated in Definition 2.2, $\mathfrak{b}_s(s) = 1$ and $\mathfrak{b}_s(s') = 0$, with $s' \neq s$. Therefore, we can declare

$$s \models \mathcal{P}_{\bowtie p}(\phi) \quad \text{iff} \quad \Pr_s\{\omega \in \mathsf{Path}_s \mid \omega \models \phi\} \bowtie p.$$

3.1 Stage One: SatPoctl*

Let $\mathcal{H} = (S, A, \Theta, B, L, \pi)$ be an HMM and Φ be a POCTL* state formula. These elements are the input of the model checking algorithm SatPoctl* shown in Figure 3.1, that computes the set $\mathsf{Sat}(\Phi) = \{s \in S \mid s \models \Phi\}$.

```
Algorithm: SatPoctl*($\mathcal{H}$, $\Phi$)
1  Identify a most deeply nested state formula $\Psi \notin AP$ of $\Phi$
2  Generate a new atomic proposition $a_\Psi$
3  $AP := AP \cup \{a_\Psi\}$
4  switch $\Psi$ do
5      case $\neg a$ where $a \in AP$
6          forall $s \in S$ such that $a \notin L(s)$ do
7              $L(s) := L(s) \cup \{a_\Psi\}$
8      case $a \vee b$ where $a, b \in AP$
9          forall $s \in S$ such that $a \in L(s) \vee b \in L(s)$ do
10             $L(s) := L(s) \cup \{a_\Psi\}$
11     case $a \wedge b$ where $a, b \in AP$
12         forall $s \in S$ such that $a \in L(s) \wedge b \in L(s)$ do
13             $L(s) := L(s) \cup \{a_\Psi\}$
14     case $\mathcal{P}_{\bowtie p}(\phi)$
15         Sat:=directApp($\mathcal{H}$, $\phi$, $\bowtie p$)
16         forall $s \in$ Sat do
17             $L(s) := L(s) \cup \{a_\Psi\}$
18     $\Phi'$ is the result of replacing this occurrence of $\Psi$ in $\Phi$ by $a_\Psi$
19 if $\Phi'$ is not an atomic proposition then
20     $\Phi := \Phi'$
21     return SatPoctl*($\mathcal{H}, \Phi$)
22 return $[s \in S \mid a_\Phi \in L(s)]$
```

Figure 3.1: The SatPoctl* algorithm, which contains the first instructions executed by de model checker.

We explain now what the algorithm does. It identifies a most deeply nested state subformula Ψ of Φ such that Ψ is not a propositional variable, i.e., $\Psi \notin AP$. A new atomic proposition a_Ψ is generated accordingly. Given the syntax depicted in Remark 3.1, for Ψ we have to consider only four cases, which are

i. Ψ is the negation of an atomic proposition.

ii. Ψ is the disjunction of two atomic propositions.

iii. Ψ is the conjunction of two atomic propositions.

iv. Ψ is the probabilistic operator.

We then find the states $s \in S$ that satisfy Ψ. This is a trivial task for the first three cases, but no trivial at all when facing the fourth case, i.e., the probabilistic operator. In order to get the states that satisfy $\mathcal{P}_{\bowtie p}(\phi)$, we invoke a subroutine called `directApp`[2], that takes as input the HMM $\mathcal{H}$ and the QOS formula $(\phi, \bowtie p)$, with ϕ the argument of $\mathcal{P}_{\bowtie p}(\phi)$ (see Section 2.3.2). This subroutine obtains the set Sat of states s such that $s \models (\phi, \bowtie p)$. According to the semantics of QOS, it is equivalent to $s \models \mathcal{P}_{\bowtie p}(\phi)$. The previously defined atomic proposition a_Ψ replaces Ψ in Φ, we named the result of this substitution as Φ'. Moreover, the label of the states satisfying Ψ is extended by a_Ψ. We set $\Phi := \Phi'$ and proceed recursively unless Φ itself was replaced by the atomic proposition a_Φ, in such case we just return the states s where a_Φ holds, that is, $a_\Phi \in L(s)$.

Example 3.2. Consider the chef's mood problem and its corresponding HMM discussed in Section 1.2. We want to find the states that satisfy the state formula $\Phi = \neg c \wedge \mathcal{P}_{>0.05}(\mathbf{X}_{\{S,F\}}d)$, where c and d are propositional formulas such that $L(H) = \{c\}$ and $L(U) = \{d\}$.

According to stage one of the model checking algorithm, we take $\Psi = \neg c$ because Ψ has to be a most nested state subformula of Φ not being a proposition variable. Only state U satisfies $\neg c$, thus its label $L(U)$ is extended with the new atomic proposition $a_{\neg c}$. The formula Φ becomes $a_{\neg c} \wedge \mathcal{P}_{>0.05}(\mathbf{X}_{\{S,F\}}d)$. Next, the recursive call is made; hence we have now $\Psi = \mathcal{P}_{>0.05}(\mathbf{X}_{\{S,F\}}d)$. The subroutine `directApp` is executed to obtain the states that make true this probabilistic operator. As we will see, the output of this subroutine is $\{H, U\}$. We extend further the label of these states to include the new atomic proposition $a_{\mathcal{P}_{>0.05}(\mathbf{X}_{\{S,F\}}d)}$. After the corresponding substitution, Φ results in $a_{\neg c} \wedge a_{\mathcal{P}_{>0.05}(\mathbf{X}_{\{S,F\}}d)}$. This conjunction of atomic propositions is satisfied by state U. The final new atomic proposition that takes this last formula Φ is a_Φ. It is solely added to the label of U. Consequently, only state U satisfies Φ.

3.2 Stage Two: Direct Approach

Stage two deals with the case encountered in line **14** of the SatPoctl* algorithm of the previous section. The `directApp` algorithm is presented here. Its inputs are the HMM $\mathcal{H}$ and the QOS formula $(\phi, \bowtie p)$. Its outputs are the states of $\mathcal{H}$ that satisfy the QOS formula; these are precisely the ones that satisfy the formula $\mathcal{P}_{\bowtie p}(\phi)$. As stated in [36], there are two ways to accomplish this goal. The first one is based on constructing an automaton in which a probabilistic reachability analysis is performed. The second one transforms the HMM $\mathcal{H}$ into a DTMC $\mathcal{D}$, and produces a QLS formula $(\phi', \bowtie p)$ from the input QOS formula $(\phi, \bowtie p)$. Remember that QLS formulas are defined in Appendix C.2. This second approach is the one considered in the implementation of our model checker; it is explained thoroughly by the next steps

[2]This subroutine is called after our choice of tackling this problem using the direct approach offered in [36].

a) From the HMM $\mathcal{H} = (S, A, \Theta, B, L, \pi)$ we define the DTMC $\mathcal{D} = (S^{\mathcal{D}}, A^{\mathcal{D}}, L^{\mathcal{D}}, \pi^{\mathcal{D}})$, where

   - $S^{\mathcal{D}} = S \times \Theta$.
   - $A^{\mathcal{D}}((s, o), (s', o')) = a_{s\,s'} \cdot b_{s'}(o')$.
   - $L^{\mathcal{D}}(s, o) = L(s) \cup \{\Omega \subseteq \Theta \,|\, o \in \Omega\}$.
   - $\pi^{\mathcal{D}}_{(s,o)} = \pi_s b_s(o)$.[3]

b) The set of atomic propositions is extended by $AP_{\mathcal{D}} = AP_{\mathcal{H}} \cup \{\Omega \,|\, \Omega \subseteq \Theta\}$.

c) We modify the OLTL formula ϕ, which is part of the input QOS formula $(\phi, \bowtie p)$, such that every occurrence of $\mathbf{X}_{\Omega}\varphi$ is replaced by $\Omega \wedge \mathbf{X}\varphi$, where Ω is a new atomic proposition just added to $AP_{\mathcal{D}}$ in the former step. Once we finish with this process, the resultant LTL formula, denoted ϕ', shall have every next operator clear of sets of observations as subscripts. Taking this ϕ' together with the original comparison operator $\bowtie$ and threshold p, the QLS formula $(\phi', \bowtie p)$ is generated.

d) We are interested in finding out whether $s \models (\phi, \bowtie p)$, for all $s \in S$. By the semantics of QOS, this is equivalent to $\Pr_s\{\omega \in \mathsf{Path}_s \,|\, \omega \models \phi\} \bowtie p$. Using the constructed DTMC $\mathcal{D}$ and QLS $(\phi', \bowtie p)$, Theorem 3.4 says that

$$\Pr_s\{\omega \in \mathsf{Path}^{\mathcal{H}}_s \,|\, \omega \models \phi\} = \sum_{o \in \Theta} \Pr_{(s,o)}\{\omega' \in \mathsf{Path}^{\mathcal{D}}_{(s,o)} \,|\, \omega' \models \phi'\}.$$

   Therefore, a method to compute the value of $\Pr_{(s,o)}\{\omega' \in \mathsf{Path}^{\mathcal{D}}_{(s,o)} \,|\, \omega' \models \phi'\}$ is needed. Fortunately, it is done with the algorithm designed by Courcoubetis *et al.* [6]. This task becomes the third stage of our model checker.

Now, we turn our attention to show why we can calculate $\Pr_s\{\omega \in \mathsf{Path}^{\mathcal{H}}_s \,|\, \omega \models \phi\}$ in terms of $\Pr_{(s,o)}\{\omega' \in \mathsf{Path}^{\mathcal{D}}_{(s,o)} \,|\, \omega' \models \phi'\}$. Our aim is to prove Theorem 3.4, that uses the next lemma, which is not present in [36].

Lemma 3.3. Let $\mathcal{H}$ be an HMM, and ϕ an OLTL formula. Moreover, let $\mathcal{D}$ be the DTMC obtained by transforming $\mathcal{H}$ according to Step a), and ϕ' be the LTL formula we get by modifying ϕ as described in Step c). Hence the next implication holds,

$$\omega \models \phi \Longrightarrow \omega' \models \phi',$$

where $\omega \in \mathsf{Path}^{\mathcal{H}}_s$ induces $\omega' \in \mathsf{Path}^{\mathcal{D}}_{(s,o_0)}$, with $s = \omega_s(0)$ and $o_0 = \omega_o(0)$.

Proof. Recall that the LTL syntax and semantics are given in Appendix C.2. The result is proved by structural induction on the syntax of OLTL (see Section 2.3.2).

[3]The initial distribution $\pi^{\mathcal{D}}_{(s,o)}$ is not considered by Zhang *et al.* in [36] but we defined it here because it is used by Theorem 3.4.

The only interesting case for the logic OLTL is when $\phi = \mathbf{X}_{o_0}\varphi$, thus

$$\begin{array}{ll}
 & \omega \models \phi \\
\Longrightarrow & \quad \{\phi = \mathbf{X}_{o_0}\varphi\} \\
 & \omega \models \mathbf{X}_{o_0}\varphi \\
\Longrightarrow & \quad \{\text{By the QOS and POCTL* semantics}\} \\
 & \omega_o(0) = o_0 \wedge \omega[1] \models \varphi \\
\Longrightarrow & \quad \{\text{Equivalently}\} \\
 & o_0 \in \{\omega_o(0)\} \wedge \omega[1] \models \varphi \\
\Longrightarrow & \quad \{\text{According to the new labelling function } L'\} \\
 & \{\omega_o(0)\} \in L'(s, o_0) \wedge \omega[1] \models \varphi \\
\Longrightarrow & \quad \begin{array}{l}\{\text{Since } \omega_o(0) = o_0 \text{ and} \\ \text{by the inductive hipothesis}\}\end{array} \\
 & \{o_0\} \in L'(s, o_0) \wedge \omega'[1] \models \varphi \\
\Longrightarrow & \quad \{\text{By the LTL semantics}\} \\
 & \omega' \models \{o_0\} \wedge \omega' \models \mathbf{X}\varphi \\
\Longrightarrow & \quad \{\text{By the LTL semantics}\} \\
 & \omega' \models \{o_0\} \wedge \mathbf{X}\varphi \\
\Longrightarrow & \quad \begin{array}{l}\{\phi = \mathbf{X}_{o_0}\varphi \text{ was transformed} \\ \text{into } \phi' = \{o_0\} \wedge \mathbf{X}\varphi\}\end{array} \\
 & \omega' \models \phi'.
\end{array}$$

∎

Theorem 3.4. *Let $\mathcal{H}$ be an HMM and $(\phi, \bowtie p)$ be a QOS formula. Let $\mathcal{D}$ be the DTMC obtained by transforming $\mathcal{H}$ according to Step a), and $(\phi', \bowtie p)$ be the QLS formula such that ϕ' results from modifying ϕ as described in Step c). Therefore, the next equation holds*

$$\Pr_s\{\omega \in \mathsf{Path}_s^{\mathcal{H}} \,|\, \omega \models \phi\} = \sum_{o \in \Theta} \Pr_{(s,o)}\{\omega' \in \mathsf{Path}_{(s,o)}^{\mathcal{D}} \,|\, \omega' \models \phi'\}.$$

Proof. The following argument is an adaptation of the one presented in [36] for a similar result involving the *automaton based approach.*

Typically, to prove results of measures over paths we have to show that they produce the same output for every single basic cylinder set. Instead, for this proof we take a family of basic cylinder sets whose paths start at s, that is, $\mathcal{F} = \bigcup_{o \in \Theta} C((s, o), \ldots, (s_k, o_k))$, where $k \in \mathbb{N}$ and every path ω in C satisfies ϕ, i.e., $\mathcal{F} \subseteq \{\omega \in \mathsf{Path}_s^{\mathcal{H}} \,|\, \omega \models \phi\}$. Depending on the observation o, it might happen that some cylinder sets are ignored by $\mathcal{F}$ since their paths might not satisfy ϕ. Observe also that the basic cylinder sets in $\mathcal{F}$ are all pairwise disjoint. By Definition A.2 we have

$$\Pr_s\Big(\bigcup_{o \in \Theta} C((s, o), \ldots, (s_k, o_k))\Big) = \sum_{o \in \Theta} \Pr_s(C((s, o), \ldots, (s_k, o_k))).$$

As we know from Section 1.2.3, the measure $\Pr_s(C((s,o),\ldots,(s_k,o_k)))$ is $\pi_s b_s(o) \cdot \prod_{i=1}^k a_{s_{i-1}\,s_i} b_{s_i}(o_i)$.

The cylinder set C induces a unique cylinder set in $\mathcal{D}$, namely $C'((s,o),\ldots,(s_k,o_k))$. Note that (s_j,o_j) is a state in $\mathcal{D}$. According to Appendix B, the probability measure $\Pr_{(s,o)}(C')$ is $\pi_{(s,o)} \prod_{i=1}^k A^{\mathcal{D}}((s_{i-1},o_{i-1}),(s_i,o_i))$. Equivalently, by construction of the $\mathcal{D}$, we have $\pi_{(s,o)} \prod_{i=1}^k a_{s_{i-1}\,s_i} b_{s_i}(o_i)$. It follows that

$$\begin{aligned}
\Pr_s\Big(\bigcup_{o\in\Theta} C((s,o),\ldots,(s_k,o_k))\Big) &= \sum_{o\in\Theta} \Pr_s(C((s,o),\ldots,(s_k,o_k))) \\
&= \sum_{o\in\Theta} \pi_s b_s(o) \cdot \prod_{i=1}^k a_{s_{i-1}\,s_i} b_{s_i}(o_i) \\
&= \sum_{o\in\Theta} \pi^{\mathcal{D}}_{(s,o)} \cdot \prod_{i=1}^k A^{\mathcal{D}}((s_{i-1},o_{i-1}),(s_i,o_i)) \\
&= \sum_{o\in\Theta} \Pr_{(s,o)}(C'((s,o),\ldots,(s_k,o_k))).
\end{aligned}$$

Furthermore, we observe that ϕ is an OLTL formula. So by Lemma 3.3 we conclude that every path $\omega' \in C'$ satisfies ϕ', i.e., $C' \subseteq \{\omega' \in \mathsf{Path}^{\mathcal{D}}_{(s,o)} \,|\, \omega' \models \phi'\}$. Therefore,

$$\Pr_s\{\omega \in \mathsf{Path}^{\mathcal{H}}_s \,|\, \omega \models \phi\} = \sum_{o\in\Theta} \Pr_{(s,o)}\{\omega' \in \mathsf{Path}^{\mathcal{D}}_{(s,o)} \,|\, \omega' \models \phi'\}.$$

■

Recall Example 3.2, where the states satisfying $\mathcal{P}_{>0.05}(\mathbf{X}_{\{S,F\}}d)$ have not yet been computed. Given the original HMM $\mathcal{H} = (S, A, \Theta, B, L, \pi)$, stage two dictates that the DTMC $\mathcal{D} = (S^{\mathcal{D}}, A^{\mathcal{D}}, L^{\mathcal{D}}, \pi^{\mathcal{D}})$ is obtained as follows:

- $S^{\mathcal{D}} = S \times \Theta = \{(H,S), (H,P), (H,F), (U,S), (U,P), (U,F)\}$.
- $A^{\mathcal{D}}((s,o),(s',o')) = a_{s\,s'} \cdot b_{s'}(o')$, that is,

$$A = \begin{array}{c} \\ (H,S) \\ (H,P) \\ (H,F) \\ (U,S) \\ (U,P) \\ (U,F) \end{array} \begin{array}{c} \begin{array}{cccccc} (H,S) & (H,P) & (H,F) & (U,S) & (U,P) & (U,F) \end{array} \\ \begin{pmatrix} 0.35 & 0.21 & 0.14 & 0.03 & 0.18 & 0.09 \\ 0.35 & 0.21 & 0.14 & 0.03 & 0.18 & 0.09 \\ 0.35 & 0.21 & 0.14 & 0.03 & 0.18 & 0.09 \\ 0.20 & 0.12 & 0.08 & 0.06 & 0.36 & 0.18 \\ 0.20 & 0.12 & 0.08 & 0.06 & 0.36 & 0.18 \\ 0.20 & 0.12 & 0.08 & 0.06 & 0.36 & 0.18 \end{pmatrix} \end{array}.$$

- $L^{\mathcal{D}}(s,o) = L(s) \cup \{\Omega \subseteq \Theta \,|\, o \in \Omega\}$. For the label of (s,o) we keep that of s, i.e., $L(s)$. Moreover, without explicitly adding $\{\Omega \subseteq \Theta \,|\, o \in \Omega\}$ to $L(s,o)$, we can know whether the subset of observations Ω is in $L(s,o)$ by simply checking whether $o \in \Omega$. This mechanism will be covered with more detail in Remark 4.1. Therefore, we have $L(H,S) = L(H,P) = L(H,F) = \{c\}$, and $L(U,S) = L(U,P) = L(U,F) = \{d, a_{\neg c}\}$.

- $\pi^{\mathcal{D}}_{(s,o)} = \pi_s b_s(o)$, that is,

$$\pi^{\mathcal{D}} = \begin{array}{c} \\ \end{array}\begin{array}{cccccc} (H,S) & (H,P) & (H,F) & (U,S) & (U,P) & (U,F) \\ \end{array}$$
$$\pi^{\mathcal{D}} = \begin{pmatrix} 0.30 & 0.18 & 0.12 & 0.04 & 0.24 & 0.12 \end{pmatrix}.$$

Additionally, the formula $\phi = \mathbf{X}_{\{S,F\}}d$ is transformed into $\phi' = \{S,F\} \wedge \mathbf{X}d$. Both the DTMC $\mathcal{D}$ and the OLTL formula $(>0.05, \phi')$ are passed to stage three, whose outcome is used to easily determine the states in $\mathcal{H}$ that satisfy Φ.

3.3 Stage Three: Courcoubetis and Yannakakis's algorithm

This section focuses on finding the value of $\Pr_{(s,o)}\{\omega' \in \mathsf{Path}^{\mathcal{D}}_{(s,o)} \,|\, \omega' \models \phi'\}$, a gap left in the previous section. We are provided with a DTMC $\mathcal{D}$ and a LTL formula ϕ'. The algorithm discussed here is taken from [6]. It transforms step-by-step both $\mathcal{D}$ and ϕ', eliminating one-by-one the temporal connectives, while preserving the probability of satisfaction of the formula. This algorithm dictates that transformations $C_{\mathbf{X}}$ and $C_{\mathcal{U}}$ must be performed for each $\mathbf{X}$ and $\mathcal{U}$ operator, respectively. The execution of each of these two constructions, described below, produces a new DTMC $\mathcal{D}'$ and a new LTL formula ψ.

The $C_{\mathbf{X}}$ transformation

The construction of $C_{\mathbf{X}}$ takes $\mathbf{X}\varphi$ as an innermost subexpression of ϕ', i.e., φ is composed of atomic propositions and Boolean connectives only. This makes easy to evaluate φ on each state of $\mathcal{D}$. Then we partition the states of the DTMC $\mathcal{D}$ into three disjoint subsets $S = S^{\mathsf{YES}} \cup S^{\mathsf{NO}} \cup S^{?}$, where:

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

Let p_{uv} denote the probability $A^{\mathcal{D}}(u,v)$, and let q_u denote the probability that $\mathbf{X}\varphi$ is satisfied starting from state u. By the above partition of the state space, we know that $q_u = 1$ if $u \in S^{\mathsf{YES}}$, $q_u = 0$ if $u \in S^{\mathsf{NO}}$. Otherwise, $q_u = \sum_v p_{uv}$, where the sum ranges over all successor states v of u satisfying formula φ.

The new DTMC $\mathcal{D}'$ has a larger state space to be constructed soon. Additionally, $\mathcal{D}'$ is defined over the new set $AP' = AP_{\mathcal{D}} \cup \{\xi\}$, where ξ[4] is a new atomic proposition.

[4]The atomic proposition ξ represents the property $\mathbf{X}\varphi$.

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 $(u, \neg\xi)$ in $\mathcal{D}'$. A state (u, ξ) satisfies all the atomic propositions that u satisfies including the new atomic proposition ξ; similarly for $(u, \neg\xi)$, except that it does not satisfy ξ.

The way we will define the transition probabilities in $\mathcal{D}'$ gives to a state (u, ξ) the following interpretation. The distribution of the paths of $\mathcal{D}'$ starting from (u, ξ), projected on the first state component, will be the same as the distribution of the paths of $\mathcal{D}$ starting from u, conditioned on the event that they satisfy $\mathbf{X}\varphi$.

Transitions of $\mathcal{D}'$
Every transition $u \to v$ in $\mathcal{D}$ implies one or two transitions in $\mathcal{D}'$. 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 at state v onward satisfies property ξ_2, conditioned on the event that in state u it satisfies property ξ_1. Let x be the transition in $\mathcal{D}$ from u to v, y be the event that a path starting at state v satisfies property ξ_2, and z be the event that in state u property ξ_1 is satisfied. Therefore, the desired probability for $(u, \xi_1) \to (v, \xi_2)$ is $P[x \wedge y \mid z]$. Now, applying the Bayes' rule (see Appendix A.1) we have

$$P[x \wedge y \mid z] = \frac{P[z \mid x \wedge y] \cdot P[x \wedge y]}{P[z]}.$$

Since the atomic events x and y are independent and we know that $P[x] = p_{u\,v}$, we can rewrite the above expression as

$$P[x \wedge y \mid z] = \frac{P[z \mid x \wedge y] \cdot p_{u\,v} \cdot P[y]}{P[z]}.$$

In more detail, we have the following cases for the transition probability of $(u, \xi_1) \to (v, \xi_2)$:

1. $u \in S^{\mathsf{YES}} \cup S^{\mathsf{NO}}$.

    1.1. $v \in S^{\mathsf{YES}} \cup S^{\mathsf{NO}}$. Then, $\mathcal{D}'$ contains a unique state (u, ξ_1) with first component u and a unique state (v, ξ_2) with first component v. We include in $\mathcal{D}'$ the transition:

    - ✓ $(u, \xi_1) \to (v, \xi_2)$ with probability $p_{u\,v}$. Since $P[y] = P[z] = P[z \mid x \wedge y] = 1$, it is true that $P[x \wedge y \mid z] = p_{u\,v}$.

    1.2. $v \in S^{?}$. Let (u, ξ_1) be the unique state of $\mathcal{D}'$ with first component u. We include two transitions:

    - ✓ $(u, \xi_1) \to (v, \xi)$ with probability $p_{u\,v} q_v$. Here $P[z] = P[z \mid x \wedge y] = 1$ and $P[y] = q_v$. Consequently, $P[x \wedge y \mid z] = p_{u\,v} q_v$
    - ✓ $(u, \xi_1) \to (v, \neg\xi)$ with probability $p_{u\,v}\overline{q_v}$, where $\overline{q_v} = 1 - q_v$. For this we also have $P[z] = P[z \mid x \wedge y] = 1$, but $P[y] = \overline{q_v}$. So $P[x \wedge y \mid z] = p_{u\,v}\overline{q_v}$.

2. $u \in S^{?}$.

  2.1. $v \in S^{\mathsf{YES}} \cup S^{\mathsf{NO}}$.

  2.1.1. v satisfies φ. We only have the transition:

  ✓ $(u, \xi) \to (v, \xi_2)$ with probability $p_{u\,v}/q_u$, where (v, ξ_2) is the unique state of $\mathcal{D}'$ with first component v. We have set such value to this transition probability because the path we are considering goes from u to v and in this case state v satisfies φ, then $P[z \mid x \wedge y] = 1$; moreover, $P[y] = 1$ and $P[z] = q_u$. Therefore, $P[x \wedge y \mid z] = p_{u\,v}/q_u$.

  2.1.2. v satisfies $\neg\varphi$. We only need the transition:

  ✓ $(u, \neg\xi) \to (v, \xi_2)$, with probability $p_{u\,v}/\overline{q_u}$. In this situation, we observe that $P[z \mid x \wedge y] = 1$ because the path we are taking goes from u to v and we are assuming that state v satisfies $\neg\varphi$. Furthermore, $P[y] = 1$ and $P[z] = \overline{q_u}$. Therefore, $P[x \wedge y \mid z] = p_{u\,v}/\overline{q_u}$.

  2.2. $v \in S^{?}$.

  2.2.1. v satisfies φ. Due to the fact that the transition goes from u to v and state v satisfies φ, we conclude that ξ holds in u, i.e., $P[z \mid x \wedge y] = 1$. We define two transitions:

  ✓ $(u, \xi) \to (v, \xi)$, with probability $p_{u\,v} q_v / q_u$. It turns out that $P[y] = q_v$ and $P[z] = q_u$. Therefore, $P[x \wedge y \mid z] = p_{u\,v} q_v / q_u$.

  ✓ $(u, \xi) \to (v, \neg\xi)$, with probability $p_{u\,v}\overline{q_v}/q_u$. Under these circumstances, we find out that $P[y] = \overline{q_v}$ and $P[z] = q_u$. Therefore, $P[x \wedge y \mid z] = p_{u\,v}\overline{q_v}/q_u$.

  2.2.2. v satisfies $\neg\varphi$. For similar reasons to previous ones, it is the case that $P[z \mid x \wedge y] = 1$. We have two transitions:

  ✓ $(u, \neg\xi) \to (v, \xi)$, with probability $p_{u\,v} q_v/\overline{q_u}$. We realise that $P[y] = q_v$ and $P[z] = \overline{q_u}$. So $P[x \wedge y \mid z] = p_{u\,v} q_v/\overline{q_u}$.

  ✓ $(u, \neg\xi) \to (v, \neg\xi)$, with probability $p_{u\,v}\overline{q_v}/\overline{q_u}$. For this case, the following probabilities hold: $P[y] = \overline{q_v}$ and $P[z] = \overline{q_u}$. Therefore, $P[z \mid x \wedge y] = p_{u\,v}\overline{q_v}/\overline{q_u}$.

Notice that we get at most two transitions in $\mathcal{D}'$. Fact that is clear if $u \in S^{\mathsf{YES}}$. Suppose now $u \in S^{?}$. When taking into account the transition from u to v, the satisfaction of ξ in u is restricted. If φ does not hold in v, then u does not satisfy ξ when going to v, e.g., a transition such as $(u, \xi) \to (v, \xi_2)$, with state v not satisfying φ, will be assigned a probability of zero since $P[z \mid x \wedge y] = 0$. Likewise for any other transition between states of $\mathcal{D}'$ not considered in the above cases.

We stress that the detailed explanation presented above is not part of [6], where only the transition probabilities formulas are given but the reasoning to reach those expressions is omitted.

Initial distribution of $\mathcal{D}'$

The interpretation of the initial distribution of $\mathcal{D}'$ given to the state (u, ξ_1) is that state u is initially chosen and the property ξ_1 is satisfied starting at u. This can be seen as the joint probability of independent events, which is just the product of the probabilities of the two events,

$$\pi^{\mathcal{D}'}_{(u,\xi_1)} = \pi^{\mathcal{D}}_u \cdot P[\xi_1 \text{ is satisfied starting at } u].$$

On the one hand, if $u \in S^{\mathsf{YES}} \cup S^{\mathsf{NO}}$, then $P[\xi_1 \text{ is satisfied starting at } u] = 1$ and $\pi^{\mathcal{D}'}_{(u,\xi_1)} = \pi^{\mathcal{D}}_u$. On the other hand, if $u \in S^?$, then $P[\xi \text{ is satisfied starting at } u] = q_u$ and $P[\neg\xi \text{ is satisfied starting at } u] = \overline{q_u}$. Furthermore, 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.

The $C_{\mathcal{U}}$ transformation

Let $\varphi_1 \mathcal{U} \varphi_2$ be an innermost temporal subexpression of ϕ', i.e., both φ_1 and φ_2 are composed of atomic propositions and Boolean connectives only; therefore, it is easy to know which states of $\mathcal{D}$ satisfy them. To obtain the new DTMC $\mathcal{D}'$, similarly to the construction $C_{\mathbf{X}}$, we partition the states of $\mathcal{D}$ in three disjoint subsets such that $S = S^{\mathsf{YES}} \cup S^{\mathsf{NO}} \cup S^?$. The interpretation of these subsets is that the paths starting at states in S^{YES} satisfy with probability one the formula $\varphi_1 \mathcal{U} \varphi_2$. If they start with states in S^{NO}, the formula $\neg(\varphi_1 \mathcal{U} \varphi_2)$ is satisfied with probability one. And if the paths start from states in $S^?$ both events will have not zero probabilities. We select the states that belong to each subset as specified by the following equations, which rely on the auxiliary algorithms ProbNo and ProbYes, which are taken from [17] and depicted in Figures 3.2 and 3.3, respectively.

$$\begin{aligned} S^{\mathsf{NO}} &= \textsc{ProbNo}(\mathsf{Sat}(\varphi_1), \mathsf{Sat}(\varphi_2)), \\ S^{\mathsf{YES}} &= \textsc{ProbYes}(\mathsf{Sat}(\varphi_1), \mathsf{Sat}(\varphi_2), S^{\mathsf{NO}}), \\ S^? &= S \setminus (S^{\mathsf{YES}} \cup S^{\mathsf{NO}}). \end{aligned}$$

where Sat is a mapping that returns the set of states in $\mathcal{D}$ that satisfy its input formula φ.

The auxiliary algorithm ProbNo first finds the states for which there is at least one path that satisfies the formula $\varphi_1 \mathcal{U} \varphi_2$. It then removes those states from S, the remaining ones are the states that satisfy $\neg(\varphi_1 \mathcal{U} \varphi_2)$ with probability exactly one.

The algorithm ProbYes behaves similar to ProbNo. It identifies the states that have at least one path that does not satisfy the formula $\varphi_1 \mathcal{U} \varphi_2$. They are set apart and, consequently, the states that are left satisfy $\varphi_1 \mathcal{U} \varphi_2$ with probability exactly one.

According to [6], the probability that $\varphi_1 \mathcal{U} \varphi_2$ is satisfied starting from state u, denoted q_u, is computed by solving the linear equation system

$$q_u = \begin{cases} 1 & \text{if } s \in S^{\mathsf{YES}} \\ 0 & \text{if } s \in S^{\mathsf{NO}} \\ \displaystyle\sum_{v \in S} p_{u\,v} \cdot q_v & \text{if } s \in S^?. \end{cases}$$

Algorithm: $\textsc{ProbNo}(\mathsf{Sat}(\varphi_1), \mathsf{Sat}(\varphi_2))$

```
1 $R := \mathsf{Sat}(\varphi_2)$
2 done := **false**
3 **while** (done = **false**) **do**
4     $R' := R \cup \{s \in \mathsf{Sat}(\varphi_1) \mid \exists s' \in R \,.\, p_{s\,s'} > 0\}$
5     **if** $R' = R$ **then** done = **true**
6     $R := R'$
7 **return** $S \setminus R$
```

Figure 3.2: The ProbNo algorithm.

Algorithm: $\textsc{ProbYes}(\mathsf{Sat}(\varphi_1), \mathsf{Sat}(\varphi_2), S^{\mathsf{NO}})$

```
1 $R := S^{\mathsf{NO}}$
2 done := **false**
3 **while** (done = **false**) **do**
4     $R' := R \cup \{s \in (\mathsf{Sat}(\varphi_1) \setminus \mathsf{Sat}(\varphi_2)) \mid \exists s' \in R \,.\, p_{s\,s'} > 0\}$
5     **if** $R' = R$ **then** done = **true**
6     $R := R'$
7 **return** $S \setminus R$
```

Figure 3.3: The ProbYes algorithm.

It is possible to rewrite the above linear equation system in the traditional matrix form $\mathbf{M} \cdot \underline{q} = \underline{b}$. We let $\mathbf{M} = \mathbf{I} - \mathbf{A}$, where $\mathbf{I}$ is the identity matrix and $\mathbf{A}$ is given by

$$\mathbf{A} = \begin{cases} p_{s\,s'} & \text{if } s \in S^? \\ 0 & \text{otherwise,} \end{cases}$$

and $\underline{b}$ is a state-indexed column vector with $\underline{b}(s)$ equal to 1 if $s \in S^{\mathsf{YES}}$, and 0 otherwise.

Note that both auxiliary algorithms are based on the computation of a fixpoint operator and will require at most $|S|$ iterations.

The new chain $\mathcal{D}'$ has a bigger state space S' and is defined over the extended set $AP' = AP_{\mathcal{D}} \cup \{\xi\}$, where ξ is a new atomic proposition. Note that ξ denotes the property $\varphi_1 \mathcal{U} \varphi_2$.

States of $\mathcal{D}'$

For each state $u \in S^{\mathsf{YES}}$ there is only one state $(u, \xi) \in \mathcal{D}'$. Similarly, for each $u \in S^{\mathsf{NO}}$ there is only one state $(u, \neg\xi) \in \mathcal{D}'$. Lastly, for each $u \in S^?$, there are two states (u, ξ) and $(u, \neg\xi)$ in $\mathcal{D}'$. The propositions satisfied in state (u, ξ) are the ones satisfied in state u plus the atomic proposition ξ, the same is true for the propositions valid in $(u, \neg\xi)$ but it does not satisfies ξ.

The interpretation of the state (u, ξ), given by the definition of the transition probability in $\mathcal{D}'$, is that the distribution of the paths of $\mathcal{D}'$ starting from (u, ξ), projected on the first state component, will be the same as the distribution of the paths of $\mathcal{D}$ starting from u conditioned on the event that they satisfy $\varphi_1 \mathcal{U} \varphi_2$.

Transitions of $\mathcal{D}'$

Every transition $u \to v$ in $\mathcal{D}$ implies one or two transitions in $\mathcal{D}'$. The transition probability of $(u, \xi_1) \to (u, \xi_2)$, with $\xi_i \in \{\xi, \neg\xi\}$ and $i \in \{1, 2\}$, is defined as being equal to the probability that $\mathcal{D}$, starting at state u, transitions next to state v and the property ξ_2 is satisfied initiating from state v, conditioned on the event that state u satisfies property ξ_1. As we have seen in the construction for $C_{\mathbf{X}}$, this transition probability corresponds to the value

$$P[x \wedge y \mid z] = \frac{P[z \mid x \wedge y] \cdot p_{u\,v} \cdot P[y]}{P[z]},$$

where x, y and z are the atomic events defined by the construction $C_{\mathbf{X}}$ when the new transition were given.

The possible transitions $(u, \xi_1) \to (v, \xi_2)$ are:

1. $u \in S^{\mathsf{YES}} \cup S^{\mathsf{NO}}$.

   1.1. $v \in S^{\mathsf{YES}} \cup S^{\mathsf{NO}}$. Then, there is a unique state (u, ξ_1) in $\mathcal{D}'$ with first component u and there is a unique state (v, ξ_2) with first component v. We include in $\mathcal{D}'$ the transition:

   - ✓ $(u, \xi_1) \to (v, \xi_2)$ with probability $p_{u\,v}$.

   1.2. $v \in S^?$. Let (u, ξ_1) be the unique state of $\mathcal{D}'$ with first component u. We include two transitions:

   - ✓ $(u, \xi_1) \to (v, \xi)$ with probability $p_{u\,v} q_v$.
   - ✓ $(u, \xi_1) \to (v, \neg\xi)$ with probability $p_{u\,v} \overline{q_v}$, where $\overline{q_v} = 1 - q_v$.

2. $u \in S^?$.

   2.1. $v \in S^{\mathsf{YES}}$.

   - ✓ $(u, \xi) \to (v, \xi)$ with probability $p_{u\,v}/q_u$, where (v, ξ) is the unique state of $\mathcal{D}'$ with first component v.

   2.2. $v \in S^{\mathsf{NO}}$.

   - ✓ $(u, \neg\xi) \to (v, \neg\xi)$ with probability $p_{u\,v}/\overline{q_u}$.

   2.3. $v \in S^?$. We define two transitions:

   - ✓ $(u, \xi) \to (v, \xi)$, with probability $p_{u\,v} q_v / q_u$.
   - ✓ $(u, \neg\xi) \to (v, \neg\xi)$, with probability $p_{u\,v} \overline{q_v} / \overline{q_u}$.

Initial distribution
If $u \in S^{\mathsf{YES}} \cup S^{\mathsf{NO}}$, there is only one state (u, ξ_1), with $\xi_1 \in \{\xi, \neg\xi\}$, whose initial probability in $\mathcal{D}'$ is $\pi^{\mathcal{D}'}_{(u,\xi_1)} = \pi^{\mathcal{D}}_u$. Nevertheless, if $u \in S^?$, the two states that are generated from it, namely (u, ξ) and $(u, \neg\xi)$, have initial probabilities $\pi^{\mathcal{D}'}_{(u,\xi)} = \pi^{\mathcal{D}}_u \cdot q_u$ and $\pi^{\mathcal{D}'}_{(u,\neg\xi)} = \pi^{\mathcal{D}}_u \cdot \overline{q_u}$, respectively.

Once either construction $C_{\mathbf{X}}$ or $C_{\mathcal{U}}$ is finished, we replace the corresponding innermost temporal subexpression in ϕ' by ξ, obtaining so the formula ψ. This completes stage three's description.

Continuing with Example 3.2, we consider the DTMC $\mathcal{D}$ and the OLTL formula $(>0.05, \phi')$ obtained by stage two. Since $\phi' = \{S, F\} \wedge \mathbf{X}d$, the construction $C_{\mathbf{X}}$ is used to reach the DTMC $\mathcal{D}'$.

First the set of states is partitioned. We realise that every state in $S^{\mathcal{D}}$ has transitions to both states satisfying d and states satisfying $\neg d$. So $u \in S^?$ for all states u in $\mathcal{D}$. Since $q_u = \sum_v p_{uv}$, where the sum ranges over all successor states v of u satisfying d, it turns out that $q_{(H,S)} = q_{(H,P)} = q_{(H,F)} = 0.3$, and $q_{(U,S)} = q_{(U,P)} = q_{(U,F)} = 0.6$.

The new set of states is $\{\big((H,S),\xi\big), \big((H,S),\neg\xi\big), \big((H,P),\xi\big), \big((H,P),\neg\xi\big), \big((H,F),\xi\big), \big((H,F),\neg\xi\big), \big((U,S),\xi\big), \big((U,S),\neg\xi\big), \big((U,P),\xi\big), \big((U,P),\neg\xi\big), \big((U,F),\xi\big), \big((U,F),\neg\xi\big)\}$.

To demonstrate how to compute the transition probabilities in $\mathcal{D}'$, we will compute the value of going from $\big((H,P),\xi\big)$ to $\big((U,F),\neg\xi\big)$. Given that both (H,P) and (U,F) are in $S^?$, and that $d \in L(U,F)$, the transition probability of $\big((H,P),\xi\big) \to \big((U,F),\neg\xi\big)$ is $p_{(H,P)\,(U,F)} \cdot \overline{q_{(U,F)}}/q_{(H,P)} = 0.09 \cdot 0.4/0.3 = 0.12$.

Also, the label of state $\big((H,S),\xi\big)$ is the one of (H,S) extended to include the atomic proposition ξ. The computation of the labels for the rest of the new states is similar.

Finally, the initial probabilities of states (u,ξ) and $(u,\neg\xi)$ are calculated by the expressions $\pi^{\mathcal{D}}_u \cdot q_u$ and $\pi^{\mathcal{D}}_u \cdot \overline{q_u}$, respectively. Consequently, $\pi^{\mathcal{D}'}$ is defined as

$$\begin{array}{ll}
\pi^{\mathcal{D}'}_{((H,S),\xi)} = 0.09, & \pi^{\mathcal{D}'}_{((H,S),\neg\xi)} = 0.21, \\
\pi^{\mathcal{D}'}_{((H,P),\xi)} = 0.054, & \pi^{\mathcal{D}'}_{((H,P),\neg\xi)} = 0.126, \\
\pi^{\mathcal{D}'}_{((H,F),\xi)} = 0.036, & \pi^{\mathcal{D}'}_{((H,F),\neg\xi)} = 0.084, \\
\pi^{\mathcal{D}'}_{((U,S),\xi)} = 0.024, & \pi^{\mathcal{D}'}_{((U,S),\neg\xi)} = 0.016, \\
\pi^{\mathcal{D}'}_{((U,P),\xi)} = 0.144, & \pi^{\mathcal{D}'}_{((U,P),\neg\xi)} = 0.096, \\
\pi^{\mathcal{D}'}_{((U,F),\xi)} = 0.072, & \pi^{\mathcal{D}'}_{((U,F),\neg\xi)} = 0.048.
\end{array}$$

To obtain the formula ψ, we replace $\mathbf{X}d$ by ξ in ϕ', that is, $\psi = \{S, F\} \wedge \xi$. We will consider $\mathcal{D}'$ and ψ again before the end of this chapter to find the states of $\mathcal{H}$ that satisfy $\mathcal{P}_{>0.05}(\mathbf{X}_{\{S,F\}}d)$.

Through Theorem 3.8 we see that the measure of paths in $\mathcal{D}$ satisfying the formula ϕ', with initial state s, is equal to the sum of two measures of paths in $\mathcal{D}'$ satisfying the formula ψ, with respective initial states (s, ξ) and $(s, \neg\xi)$. Some observations and lemmas are needed prior to that result.

Let $\omega = \omega_0, \ldots, \omega_m \in \mathsf{Path}^{fin,\mathcal{D}}_{\omega_0}$. It induces the path $\sigma = \sigma_0, \ldots, \sigma_m \in \mathsf{Path}^{fin,\mathcal{D}'}_{\sigma_0}$, such that $\mathsf{fst}(\sigma_i) = \omega_i$, $\forall i \in \{0, \ldots, m\}$. Observe that if (v, ξ_l), with $\xi_l \in \{\xi, \neg\xi\}$, is a state in the induced path σ and u is an immediate predecessor of v in ω, then (v, ξ_l) has exactly one immediate predecessor in σ with first component u.

- If $u \in S^{\mathsf{YES}} \cup S^{\mathsf{NO}}$, then there is only one state with first component u in $\mathcal{D}'$.
- If $u \in S^{?}$, then for the construction $C_{\mathcal{U}}$, the immediate predecessor of (v, ξ_l) is (u, ξ_l), i.e., it agrees in the second component. For the construction $C_{\mathbf{X}}$, the immediate predecessor of (v, ξ_l) is (u, ξ) if v satisfies φ, and $(u, \neg\xi)$ if v satisfies $\neg\varphi$.

Hence, having final state $\sigma_m = (\omega_m, \xi_m)$ and proceeding backwards, we see that the set $\mathsf{Path}^{fin,\mathcal{D}'}_{\sigma_0}$ has a unique path σ which ends at σ_m and has projection $\omega_0, \ldots, \omega_m$.

The next result is mentioned in [6] but not proved as we do here.

Lemma 3.5. Let $\omega = \omega_0, \ldots, \omega_m$ be an element of $\mathsf{Path}^{fin,\mathcal{D}}_{\omega_0}$, inducing the unique path $\sigma = \sigma_0, \ldots, \sigma_m \in \mathsf{Path}^{fin,\mathcal{D}'}_{\sigma_0}$, with $\sigma_m = (\omega_m, \xi_m)$. Let $C(\omega)$ and $C'(\sigma)$ be their respective cylinder sets. If $\xi_m = \xi$, then $\Pr_{\sigma_0}(C'(\sigma)) = \Pr_{\omega_0}(C(\omega)) \cdot q_{\omega_m}$, and if $\xi_m = \neg\xi$, then $\Pr_{\sigma_0}(C'(\sigma)) = \Pr_{\omega_0}(C(\omega)) \cdot \overline{q}_{\omega_m}$.

Proof. We will discuss here the case when the construction applied is $C_{\mathcal{U}}$. For $C_{\mathbf{X}}$, the argument is similar.

We prove this lemma by induction on the length m of the path ω.

Base Case. Take $m = 0$, we recognise three different situations. First, if $\omega_0 \in S^{\mathsf{YES}}$, then $\xi_0 = \xi$ and $q_{\omega_0} = 1$. So, by the initial distribution of $\mathcal{D}'$, we have $\Pr_{\sigma_0}(C'(\sigma)) = \pi^{\mathcal{D}'}_{\sigma_0} = \pi^{\mathcal{D}}_{\omega_0} = \Pr_{\omega_0}(C(\omega)) \cdot q_{\omega_0}$. Secondly, if $\omega_0 \in S^{\mathsf{NO}}$, then the reasoning is analogous. Thirdly, if $\omega_0 \in S^{?}$ and $\xi_0 = \xi$, then $\Pr_{\sigma_0}(C'(\sigma)) = \pi^{\mathcal{D}}_{\omega_0} \cdot q_{\omega_0} = \Pr_{\omega_0}(C(\omega)) \cdot q_{\omega_0}$. The argument is similar when $\xi_0 = \neg\xi$.

Induction step. We assume that the lemma holds for $m = k$. Let us prove that the lemma is also true for $m = k + 1$. By the definition of probability measure for DTMCs given in Appendix B, we have $\Pr_{\sigma_0}(C'(\sigma_0, \ldots, \sigma_{k+1})) = \Pr_{\sigma_0}(C'(\sigma_0, \ldots, \sigma_k)) \cdot A^{\mathcal{D}'}(\sigma_k, \sigma_{k+1})$. Here we use the induction hypothesis and see that the last expression is either $\Pr_{\omega_0}(C(\omega_0, \ldots, \omega_k)) \cdot q_{\omega_k} \cdot A^{\mathcal{D}'}(\sigma_k, \sigma_{k+1})$ if $\sigma_k = (\omega_k, \xi)$ or $\Pr_{\omega_0}(C(\omega_0, \ldots, \omega_k)) \cdot \overline{q}_{\omega_k} \cdot A^{\mathcal{D}'}(\sigma_k, \sigma_{k+1})$ if $\sigma_k = (\omega_k, \neg\xi)$.

For the case when $\omega_k \in S^{\mathsf{YES}}$ (if $\omega_k \in S^{\mathsf{NO}}$ we follow much of these same lines) we have $q_{\omega_k} = 1$. Furthermore, we consider three possibilities for ω_{k+1}. If $\omega_{k+1} \in S^{\mathsf{YES}}$, then $q_{\omega_{k+1}} = 1$ and $A^{\mathcal{D}'}(\sigma_k, \sigma_{k+1}) = p_{\omega_k \omega_{k+1}}$. Therefore, we have

$$\begin{aligned}\Pr_{\omega_0}(C(\omega_0, \ldots, \omega_k)) \cdot q_{\omega_k} \cdot A^{\mathcal{D}'}(\sigma_k, \sigma_{k+1}) &= \Pr_{\omega_0}(C(\omega_0, \ldots, \omega_k)) \cdot 1 \cdot p_{\omega_k \omega_{k+1}} \\ &= \Pr_{\omega_0}(C(\omega_0, \ldots, \omega_{k+1})) \cdot q_{\omega_{k+1}}.\end{aligned}$$

The option when $\omega_{k+1} \in S^{\mathsf{NO}}$ is similar. Moreover, if $\omega_{k+1} \in S^{?}$ and $\sigma_{k+1} = (\omega_{k+1}, \xi)$, then $A^{\mathcal{D}'}(\sigma_k, \sigma_{k+1}) = p_{\omega_k\,\omega_{k+1}} q_{\omega_{k+1}}$. Thus, we compute the sought probability as

$$\begin{aligned}\Pr_{\omega_0}(C(\omega_0, \ldots, \omega_k)) \cdot\ q_{\omega_k} \cdot A^{\mathcal{D}'}(\sigma_k, \sigma_{k+1}) &= \Pr_{\omega_0}(C(\omega_0, \ldots, \omega_k)) \cdot 1 \cdot p_{\omega_k\,\omega_{k+1}} q_{\omega_{k+1}} \\ &= \Pr_{\omega_0}(C(\omega_0, \ldots, \omega_{k+1})) \cdot\ q_{\omega_{k+1}}.\end{aligned}$$

When $\sigma_{k+1} = (\omega_{k+1}, \neg\xi)$, then we proceed in a similar fashion.

Finally, if $\omega_k \in S^?$, we should analyse three cases for ω_{k+1} as before. Nonetheless, we take only one case since the others are analogous. When $\omega_{k+1} \in S^{\mathsf{YES}}$, then $q_{\omega_{k+1}} = 1$ and $A^{\mathcal{D}'}(\sigma_k, \sigma_{k+1}) = p_{\omega_k\,\omega_{k+1}}/q_{\omega_k}$. Consequently,

$$\begin{aligned}\Pr_{\omega_0}(C(\omega_0, \ldots, \omega_k)) \cdot\ q_{\omega_k} \cdot A^{\mathcal{D}'}(\sigma_k, \sigma_{k+1}) &= \Pr_{\omega_0}(C(\omega_0, \ldots, \omega_k)) \cdot\ q_{\omega_k} \cdot p_{\omega_k\,\omega_{k+1}}/q_{\omega_k} \\ &= \Pr_{\omega_0}(C(\omega_0, \ldots, \omega_k)) \cdot p_{\omega_k\,\omega_{k+1}} \\ &= \Pr_{\omega_0}(C(\omega_0, \ldots, \omega_{k+1})) \cdot\ q_{\omega_{k+1}}.\end{aligned}$$

∎

The next lemma is a particular case of a similar result appearing in [6]. Here we focused on the probability of satisfaction of specific states rather than the overall value.

Lemma 3.6. Let $\omega = s, s_1, \ldots, s_m \in \mathsf{Path}_s^{fin,\mathcal{D}}$. It induces one or two finite paths, $\sigma = (s, \xi_0), (s_1, \xi_1), \ldots, (s_m, \xi_m) \in \mathsf{Path}_{(s,\xi_0)}^{fin,\mathcal{D}'}$, where $\xi_k \in \{\xi, \neg\xi\}$, for all $0 \leq k \leq m$. Let $C(\omega)$ and $C'(\sigma)$ be their respective cylinder sets. Then,

$$\Pr_s\big(C(\omega)\big) = \sum_{\xi_i \in \{\xi, \neg\xi\}} \Pr_{(s,\xi_i)}\big(C'(\sigma)\big).$$

Proof. As we have seen, if $s_m \in S^{\mathsf{YES}} \cup S^{\mathsf{NO}}$, then there is only one possible induced path σ. Suppose that $s_m \in S^{\mathsf{YES}}$ and σ starts at state (s, ξ), then $q_{s_m} = 1$ and by Lemma 3.5

$$\begin{aligned}\Pr_s(C(\omega)) &= \Pr_s(C(\omega)) \cdot q_{s_m} \\ &= \Pr_{(s,\xi)}(C'(\sigma)).\end{aligned}$$

The lemma holds since the other operand in the summation is zero, because there is no induced path σ with initial state $(s, \neg\xi)$.

If $s_m \in S^?$, there are two states for s_m, namely (s_m, ξ) and $(s_m, \neg\xi)$, and therefore two possible paths σ induced by ω. It could have happened that both of these two paths share the same initial state, say $(s, \neg\xi)$. By Lemma 3.5 and Definition A.2, it follows that

$$\begin{aligned}\Pr_s(C(\omega)) &= \Pr_s(C(\omega)) \cdot (q_{s_m} + \overline{q}_{s_m}) \\ &= \Pr_s(C(\omega)) \cdot q_{s_m} + \Pr_s(C(\omega)) \cdot \overline{q}_{s_m} \\ &= \Pr_{(s,\neg\xi)}(C'(\sigma_0, \ldots, \sigma_m = (s_m, \xi))) + \Pr_{(s,\neg\xi)}(\sigma_0, \ldots, \sigma_m = (s_m, \neg\xi))) \\ &= \Pr_{(s,\neg\xi)}(C'(\sigma_0, \ldots, \sigma_m = (s_m, \xi)) \cup C'(\sigma_0, \ldots, \sigma_m = (s_m, \neg\xi))),\end{aligned}$$

clearly, $q_{s_m} + \overline{q}_{s_m} = 1$. Again, the lemma holds since the other term in the summation is zero.

But it could also have happened that these two paths induced by ω originate from different starting points. Suppose that we have one path starting at $(s, \neg\xi)$ with final state (s_m, ξ), and a second path starting at (s, ξ) with final state $(s_m, \neg\xi)$. By Lemma 3.5, we conclude,

$$\begin{aligned}
\Pr_s(C(\omega)) &= \Pr_s(C(\omega)) \cdot (q_{s_m} + \overline{q}_{s_m}) \\
&= \Pr_s(C(\omega)) \cdot q_{s_m} + \Pr_s(C(\omega)) \cdot \overline{q}_{s_m} \\
&= \Pr_{(s,\neg\xi)}(C'(\sigma_0, \ldots, \sigma_m = (s_m, \xi))) + \Pr_{(s,\xi)}(C'(\sigma_0, \ldots, \sigma_m = (s_m, \neg\xi))).
\end{aligned}$$

All the remaining cases are solved analogously. ∎

Lemma 3.7. With probability one, a path $\sigma \in \mathsf{Path}^{\mathcal{D}'}$ satisfies the following property: $\forall t \geq 0$, ξ is an atomic proposition satisfied at state σ_t iff the suffix $\sigma[t] = \sigma_t, \sigma_{t+1}, \ldots$ satisfies the path formula $\varphi_1 \mathcal{U} \varphi_2$ (respectively $\mathbf{X}\varphi$). In other words, $\varphi_1 \mathcal{U} \varphi_2 \equiv \xi$ (respectively $\mathbf{X}\varphi \equiv \xi$) holds with probability one at all times in all paths in $\mathsf{Path}^{\mathcal{D}'}$.

Proof. The proof presented here is taken from the paper written by Courcoubetis *et al.*, [6]. In it only the until operator is considered; nonetheless, if the next operator were taken, a similar argument would be used.

It suffices to show that starting at any state of the form (u, ξ) a path σ satisfies with probability one the specification $\varphi_1 \mathcal{U} \varphi_2$. Likewise, if σ starts from a state $(u, \neg\xi)$, it will satisfy with probability one the negation of the previous specification. We examine the three possible cases for the initial state of σ.

a) $(u, \neg\xi)$, with $u \in S^{\mathsf{NO}}$. Looking at the DTMC $\mathcal{D}'$ as a directed graph, we obtain its subgraph G_1 formed with the states whose first component is in S^{NO}. We observe that all transitions out of G_1 occur on states of G_1 satisfying $\neg\varphi_1$ and $\neg\varphi_2$. Moreover, we know that all states in G_1 satisfy $\neg\varphi_2$. If the initial state $(u, \neg\xi)$ satisfies also $\neg\varphi_1$, the formula $\neg(\varphi_1 \mathcal{U} \varphi_2)$ is satisfied with probability one trivially. If the initial state $(u, \neg\xi)$ satisfies φ_1, then with probability one any path of $\mathcal{D}'$ will either remain in states of G_1 that satisfy φ_1 and $\neg\varphi_2$ or eventually reach a state satisfying $\neg\varphi_1$ and $\neg\varphi_2$. In either case σ satisfies $\neg(\varphi_1 \mathcal{U} \varphi_2)$.

b) (u, ξ), with $u \in S^{\mathsf{YES}}$. We consider the subgraph G_2 of $\mathcal{D}'$ induced on the states with first component belonging to S^{YES}. For this subgraph G_2, all transitions out of it occur on states of G_2 satisfying φ_2. Furthermore, every state in G_2 satisfy φ_1 and $\neg\varphi_2$, or φ_2. An important remark is that there is no strongly connected component[5] of G_2, without transitions coming out of it, consisting solely of states satisfying both φ_1 and $\neg\varphi_2$ because this component would have to be part of G_1. If the path's initial state (u, ξ) satisfies φ_2, with probability one the formula $\varphi_1 \mathcal{U} \varphi_2$ is satisfied trivially. If $\sigma \in \mathsf{Path}^{\mathcal{D}'}$ starts at a state (u, ξ) satisfying φ_1 and $\neg\varphi_2$, then with probability one the path σ will reach a state satisfying φ_2, thus it will satisfy $\varphi_1 \mathcal{U} \varphi_2$.

[5]A *strongly connected component* of a graph is a maximal subgraph where every node can reach every other node.

c) (u, ξ) or $(u, \neg\xi)$, with $u \in S^?$. Consider the subgraphs G_3 and G_4 of $\mathcal{D}'$ induced on the states (u, ξ) and $(u, \neg\xi)$, respectively, where $u \in S^?$. These subgraphs do not have strongly connected components of $\mathcal{D}'$ without transitions out of them since this components would be part of G_1. Moreover, all transition out of G_3 are into state in G_2 and all transitions out of G_4 are into states in G_1. If σ starts at a state (u, ξ) in G_3, then it will reach a state in G_2 with all previous states in G_3. Since the states in G_3 satisfy both φ_1 and $\neg\varphi_2$, and once the path enters into G_2 it certainly will satisfy $\varphi_1 \mathcal{U} \varphi_2$, it follows that σ satisfies with probability one the formula $\varphi_1 \mathcal{U} \varphi_2$. With a similar argument we can conclude that any path starting at a state $(u, \neg\xi)$ in G_4 will satisfy $\neg(\varphi_1 \mathcal{U} \varphi_2)$ with probability one.

∎

Theorem 3.8. *Let $\mathcal{D}$ be a DTMC, ϕ' be an LTL formula, and α be an innermost temporal subformula of ϕ'. Let $\mathcal{D}'$ be the DTMC resulting from applying either transformation $C_{\mathcal{U}}$ or $C_{\mathbf{X}}$ to $\mathcal{D}$, and ψ the formula resulting from replacing α by ξ in ϕ'. Then*

$$\Pr_s\{\omega \in \mathsf{Path}_s^{\mathcal{D}} \mid \omega \models \phi'\} = \sum_{\xi_i \in \{\xi, \neg\xi\}} \Pr_{(s,\xi_i)}\{\sigma \in \mathsf{Path}_{(s,\xi_i)}^{\mathcal{D}'} \mid \sigma \models \psi\}.$$

Proof. By Lemma 3.6, it is true that

$$\Pr_s\{\omega \in \mathsf{Path}_s^{\mathcal{D}} \mid \omega \models \phi'\} = \sum_{\xi_i \in \{\xi, \neg\xi\}} \Pr_{(s,\xi_i)}\{\sigma \in \mathsf{Path}_{(s,\xi_i)}^{\mathcal{D}'} \mid \sigma \models \phi'\}.$$

Notice that ω induces the path σ, so by construction the state σ_t satisfies the atomic propositions valid in ω_t at each instant t; therefore, $\omega \models \phi'$ implies $\sigma \models \phi'$. Now, by Lemma 3.7, we have $\varphi_1 \mathcal{U} \varphi_2 \equiv \xi$ ($\mathbf{X}\varphi \equiv \xi$), thus $\phi' \equiv \psi$. In consequence

$$\sum_{\xi_i \in \{\xi, \neg\xi\}} \Pr_{(s,\xi_i)}\{\sigma \in \mathsf{Path}_{(s,\xi_i)}^{\mathcal{D}'} \mid \sigma \models \phi'\} = \sum_{\xi_i \in \{\xi, \neg\xi\}} \Pr_{(s,\xi_i)}\{\sigma \in \mathsf{Path}_{(s,\xi_i)}^{\mathcal{D}'} \mid \sigma \models \psi\}.$$

∎

Recall that our goal is to find $\Pr_{(s,o)}\{\omega' \in \mathsf{Path}_{(s,o)}^{\mathcal{D}} \mid \omega' \models \phi'\}$. If the ϕ' has k temporal operators, we can compute this measure as follows: apply k times the appropriate transformations $C_{\mathbf{X}}$ or $C_{\mathcal{U}}$ to get the sequences $\mathcal{D}^1, \ldots, \mathcal{D}^k$ and $\psi^1, \ldots, \psi^k$ of DTMCs and LTL formulas[6], respectively, where ψ^k is a simple propositional formula. Then by applying Theorem 3.8 repeatedly, the formula to compute the desired measure is

$$\Pr_{(s,o)}\{\omega' \in \mathsf{Path}_{(s,o)}^{\mathcal{D}} \mid \omega' \models \phi'\} = \sum_{\substack{\xi_{i_1} \in \{\xi_1, \neg\xi_1\} \\ \vdots \\ \xi_{i_k} \in \{\xi_k, \neg\xi_k\}}} \Pr_{\underbrace{(\ldots((s,o),\xi_{i_1}),\ldots,\xi_{i_k})}_{\sigma_0}}\{\sigma \in \mathsf{Path}_{\sigma_0}^{\mathcal{D}^k} \mid \sigma \models \psi^k\}.$$

[6]The initial elements in these sequences are such that $\mathcal{D}' = \mathcal{D}^1$ and $\psi = \psi^1$.

Since ψ^k is a propositional formula, the right-hand side of the previous equality is just the sum of the initial distribution in $\mathcal{D}^k$ of the states $(\ldots((s,o),\xi_{i_1}),\ldots,\xi_{i_k})$ that satisfy ψ^k.

With all the work done so far we can bring on the crucial result that allows to compute the probability that an HMM state satisfies a probabilistic formula. Furthermore, in [36] the authors conclude their calculation of the model checker with Theorem 3.4, yet we decided to give the explicit calculation that integrates the work presented in [6]. Thus, we arrive at the corollary below.

Corollary 3.9. *Let $\mathcal{H} = (S, A, \Theta, B, L, \pi)$ be an HMM, ϕ be an OLTL path formula, and s be a state in S. The next property holds*

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

where $\mathcal{D}^k$ and ψ^k are output by stage three, which takes as input the DTMC $\mathcal{D}$ and the LTL formula ϕ'. These last values are returned by stage two, when it is provided with $\mathcal{H}$ and ϕ.

Proof. It follows immediately from all the previous results and transformations in this chapter. ∎

To complete Example 3.2, we spot that the only states satisfying $\psi = \{S, F\} \wedge \xi$, where state H is involved, are $\big((H, S), \xi\big)$ and $\big((H, F), \xi\big)$. Thus the sum of their respective initial probabilities is $0.09 + 0.036 = 0.126$, which is greater than 0.05. Similarly, the sum of the initial probabilities of the states satisfying ψ, where state U is involved, is 0.096. This number is also greater than 0.05. Therefore, both H and U satisfy $\mathcal{P}_{>0.05}(\mathbf{X}_{\{S,F\}}d)$.

The reader may have noticed the missing transformation for the *bounded until* temporal operator. It is also missing in [6], where the other two constructions are found. However, as a final remark of this section, once the state set partition ($S = S^{\mathsf{YES}} \cup S^{\mathsf{NO}} \cup S^?$) is obtained the construction $C_{\mathcal{U}^{\leq n}}$ follows the same lines of the $C_{\mathcal{U}}$ transformation, regarding the new set of states, transition probabilities and initial distribution. Furthermore, the theorem, lemmas and results hold for this new construction $C_{\mathcal{U}^{\leq n}}$ and the proofs can be easily extended to include this transformation too.

The difficulty consists in finding the partition $S = S^{\mathsf{YES}} \cup S^{\mathsf{NO}} \cup S^?$, where the paths with initial state in S^{YES} satisfy with probability one the formula $\varphi_1\mathcal{U}^{\leq n}\varphi_2$; whereas the paths starting at states belonging to S^{NO} satisfy with probability one the formula $\neg(\varphi_1\mathcal{U}^{\leq n}\varphi_2)$; and the paths that start at the remaining states, i.e., members of $S^?$, satisfy both properties with non zero probability. The way we find this partition is by calculating first the probabilities q_u. Recall that for each state $u \in S$, q_u is the probability that $\varphi_1\mathcal{U}^{\leq n}\varphi_2$ is satisfied having u as initial state. Since both φ_1 and φ_2

are propositional formulas we can find the sets $\mathsf{Sat}(\varphi_1)$ and $\mathsf{Sat}(\varphi_2)$, with them we can compute the desired values q_u in accordance with the method presented in [17]. In there, a naive state set partition is first computed as

$$X^{\mathsf{NO}} = S \setminus (\mathsf{Sat}(\varphi_1) \cup \mathsf{Sat}(\varphi_2)), \qquad X^{\mathsf{YES}} = \mathsf{Sat}(\varphi_2), \qquad X^{?} = S \setminus (X^{\mathsf{NO}} \cup X^{\mathsf{YES}}).$$

The probability that $\phi_1 \mathcal{U}^{\leq n} \phi_2$ is satisfied starting at u will be denoted $q_u(\varphi_1 \mathcal{U}^{\leq n} \varphi_2)$ instead of q_u.

Trivially, the states in $u \in X^{\mathsf{YES}}$ have $q_u(\varphi_1 \mathcal{U}^{\leq n} \varphi_2) = 1$; if $u \in X^{\mathsf{NO}}$, then $q_u(\varphi_1 \mathcal{U}^{\leq n} \varphi_2) = 0$. For the states in $X^{?}$, we find the value $q_u(\varphi_1 \mathcal{U}^{\leq n} \varphi_2)$ through the recursive definition,

$$q_u(\varphi_1 \mathcal{U}^{\leq n} \varphi_2) = \begin{cases} 0 & \text{if } n = 0 \\ \displaystyle\sum_{v \in S} p_{u\,v} \cdot q_v(\varphi_1 \mathcal{U}^{\leq n-1} \varphi_2) & \text{if } n \geq 1. \end{cases}$$

Moreover, we defined an auxiliary matrix $\mathbf{A}$, where

$$\mathbf{A}(u, v) = \begin{cases} p_{u\,v} & \text{if } u \in X^{?} \\ 1 & \text{if } u \in X^{\mathsf{YES}} \text{ and } u = v \\ 0 & \text{otherwise.} \end{cases}$$

Using the matrix $\mathbf{A}$, the required probability can be expressed as an n matrix-vector multiplication. For $n = 0$, we define the vector $\underline{q(\varphi_1 \mathcal{U}^{\leq 0} \varphi_2)}(u) = 1$ if $u \in X^{\mathsf{YES}}$ and 0 otherwise. For $n > 0$

$$\underline{q(\varphi_1 \mathcal{U}^{\leq n} \varphi_2)} = \mathbf{A} \cdot \underline{q(\varphi_1 \mathcal{U}^{\leq n-1} \varphi_2)}.$$

At the end of the computation, the states u with $q_u(\varphi_1 \mathcal{U}^{\leq n} \varphi_2) = 1$ are in S^{YES}; likewise, if $q_u(\varphi_1 \mathcal{U}^{\leq n} \varphi_2) = 0$, then $u \in S^{\mathsf{NO}}$; else $u \in S^{?}$.

3.4 Time Complexity

Let $\mathcal{H} = (S, A, \Theta, B, L, \pi)$ be an HMM and Φ a POCTL* state formula. In this section we analyse the time complexity of the POCTL* model checking algorithm executed on $\mathcal{H}$ and Φ. We will follow the ideas provided in [6, 36]. We define the size of a formula, denoted by $|\Phi|$, as its number of Boolean and temporal connectives.

Clearly, the complexity of stage one is linear in the size of the formula Φ because the algorithm in Figure 3.1 recursively takes all the subformulas of Φ.

If the probabilistic operator is encountered, then stage two is performed. In the constructed DTMC $\mathcal{D} = (S^{\mathcal{D}}, A^{\mathcal{D}}, L^{\mathcal{D}}, \pi^{\mathcal{D}})$, the size of the $A^{\mathcal{D}}$ can be, in the worst case, $O(|S|^2|\Theta|^2)$. Note that the size of the transformed Φ is at most twice that of the original Φ. Although, the number of temporal operators remains at most $|\Phi|$.

Finally, we examine the time needed to run stage three. This stage aims at forming the DTMC $\mathcal{D}'$ by applying the construction $C_{\mathbf{X}}$, $C_{\mathcal{U}}$ or $C_{\mathcal{U}^{\leq n}}$ to $\mathcal{D}$. The main component of these constructions is the states partition into the sets S^{YES}, S^{NO} and $S^{?}$, as well as the computation of the probability values q_u for each state $u \in S$. Once these two tasks are finished, the rest of the construction is straightforward. It turns out that for $C_{\mathbf{X}}$ the states partition and the values q_u are both found in time $O(|S^{\mathcal{D}}|^2)$. For $C_{\mathcal{U}}$ the states partition takes time $O(|S^{\mathcal{D}}|^2)$. Note that to find the values q_u it has to solve a linear equation system. Here the Gaussian elimination method[7] can be used, which incurs a time complexity of $O(|S^{\mathcal{D}}|^3)$ at most. Additionally, in the construction $C_{\mathcal{U}^{\leq n}}$ the states partition is achieved by first computing q_u, which requires n matrix-vector multiplications. Thus the running time of $C_{\mathcal{U}^{\leq n}}$ is $O(n\,|S^{\mathcal{D}}|^2)$. Given that the elimination of each temporal operator at most doubles the number of states and transitions, the time needed to construct the final DTMC $\mathcal{D}^k$ is $O(2^{|\Phi|}\,|S^{\mathcal{D}}|^3)$, that is, $O(2^{|\Phi|}\,|S|^3|\Theta|^3)$.

The model checking algorithm for the logic POCTL* was presented in this chapter. We saw that it can be split in three main stages. Furthermore, some important claims were stated and proven. Noticeably, in order to be adequate to our particular conditions, these claims shown here have minor modifications to the ones presented in the original papers. Finally, we have analysed the time complexity of the POCTL* model checking algorithm. The next chapter deals with the model checker's implementation in HASKELL. First, we will choose a convenient representation for the HMM elements as well as for the POCTL* formulas. Then, the three stages will be coded and some strategies to cope with theoretical issues will be offered.

[7]In Chapter 4 we see that the SMT solver *Z3* is part of our implementation. This tool solves linear equation systems by the Gaussian elimination method, as expressed via a direct e-mail by Nikolaj Bjorner, one of its developers.

Chapter 4

Implementation of a Model Checker

Up to this point we have studied in detail the model checking algorithm for the POCTL* logic, this chapter exhibits its implementation in the functional programming language HASKELL. This programming language was chosen to code the model checker because it allows the programmer to work in a high-level abstract layer, neither worrying about memory allocation, object creation nor classes handling. It also manages recursion efficiently, better that many other programming languages, this is a vital feature since recursive calls are made continuously throughout the implementation, as we shall see.

We will start this chapter discussing how HMMs and DTMCs are designed in HASKELL. Also we will offer adequate representations to the POCTL* and LTL syntactic rules. Afterwards, we will describe the implementation of each of the three stages constituting the algorithm. Moreover, we will explain how we managed to solve some difficulties encountered while coding; in them a few new additional features to the original model checker are also considered and studied.

Additionally, the model checker code is organised in different files that are submitted together with this thesis and can be accessed if more details and documentation of the implementation are needed.

4.1 Coding HMMs and DTMCs in HASKELL

Recall Definition 1.2 of an HMM. Succinctly, an HMM is a tuple $\mathcal{H} = (S, A, \Theta, B, L, \pi)$. It is straightforward to associate this tuple with a record value. We represent S, A, B, L and π as arrays; finally, Θ is encoded as a list. Hence, in code this definition is

```haskell
data HMM = HMM {
  statesHMM :: Array Int Int,           -- States
  pTransHMM :: Array (Int, Int) Double,-- State Transition Matrix
  labelFHMM :: Array Int [String],      -- Labelling Function
  obsHMM    :: [Int],                   -- Set of observation
  pObsHMM   :: Array (Int, Int) Double,-- Observation Probability Matrix
  initDiHMM :: Array Int Double         -- Initial Distribution Matrix
  } deriving (Show)
```

Observe how we have chosen `Int` values to become the states of the HMM; this type also is used to code the observations. We see that every probability matrix entry is taken as a `Double` value. The labels with the atomic propositions holding in their associated states are described as lists of `String` values.

Since DTMCs are tuples too, this same idea of working with records is considered when designing the data type for them. As stated in Definition 1.1, a DTMC is a tuple $\mathcal{D} = (S, A, L, \pi)$. Then, we have the coding

```
data DTMC a = DTMC {
  statesDTMC :: Array Int a,             -- States
  pTransDTMC :: Array (Int, Int) Double,-- State Transition Matrix
  labelFDTMC :: Array Int [String],      -- Labelling Function
  initDiDTMC :: Array Int Double         -- Initial Distribution Matrix
  } deriving (Show)
```

Interestingly, the above type variable `a` plays a prominent role in this definition, since in stage two of the model checking algorithm `a` is instantiated to be a pair of values. In stage three `a` becomes a recursive data type in order to handle the various transformations $C_{\mathbf{X}}$, $C_{\mathcal{U}}$ and $C_{\mathcal{U}^{\leq n}}$. Furthermore, the model checker requires the type variable `a` to instantiate the class `Eq`[8].

4.2 POCTL* and LTL Data Types

Another basic component of our model checker is the Haskell representation of the rules that determine how to construct well-formed POCTL* expressions. This corresponds to the grammar shown in Remark 3.1, where two grammar categories are defined, namely state and path formulas. Once again, we exploit the functional paradigm by which Haskell is characterised, since the data type we devised for the POCTL* formulas makes it clear that it derives from their formal definition, i.e.,

```
data POCTL  = VerdadP | FalsoP | AtomP String | NoP POCTL
            | OP POCTL POCTL | YP POCTL POCTL | Prob String Double PathF

data PathF  = FormP POCTL | NoPath PathF | OPath PathF PathF
            | YPath PathF PathF | NextPath [Int] PathF
            | UntilBPath Int PathF PathF | UntilPath PathF PathF
```

Let us remember the way the model checking algorithm deals with most deeply nested state subformula when it matches the probabilistic operator $\mathcal{P}_{\bowtie p}(\phi)$, where ϕ is an OLTL formula. In this situation the stage two is performed. It particularly modifies ϕ to get an LTL formula by removing the set of observations Ω attached to the next operator $\mathbf{X}_{\Omega}\psi$, and then generating the conjunction $\Omega \wedge \mathbf{X}\psi$. Consequently, the set of

[8]The `Eq` class consists of the methods `(==)` and `(/=)`, to instantiate `Eq` it suffices to defining either one of them.

atomic propositions $AP_{\mathcal{D}}$, over which this resulting LTL formula is defined, takes the original atomic propositions and considers also any set of observations Ω as an atomic proposition. Thus, we have to code these two possibilities of elements belonging to $AP_{\mathcal{D}}$. We achieve this by defining a data type that can be either a simple atomic proposition or a set of observations, as appears next

```
data AtomElem = Simple String | ObSet [Int]
```

As we can see, a simple atomic proposition is just a value of type `String`. Observe that the `POCTL` definition shown above similarly takes strings as atomic propositions.

To implement the syntax of the LTL formula that is produced by stage two of the model checker, we have the data type

```
data LTL  = Verdad | Falso | Atom AtomElem | No LTL | O LTL LTL
    | Y LTL LTL | Next LTL | UntilB Int LTL LTL | Until LTL LTL
```

This data type does match the definition of LTL given in Appendix C.2.

4.3 Implementing Stage One

The implementation of stage one is coded in a module named `ModelChecker.hs`. The main function of this module is `satPOCTL` that corresponds to the algorithm in Figure 3.1, whose inputs are an HMM and a POCTL* formula, which are coded according to the two previous sections, and returns the list of HMM's states satisfying the specification. We define this function using pattern matching of the data type `POCTL`. There are three base cases `VerdadP`, `FalsoP` and `AtomP s`. For example, if we take the latter base case we have

```
satPOCTL hmm (AtomP s) = [(statesHMM hmm)!ind | ind<-indices lab,
                                                  elem s (lab!ind)]
    where
      lab = labelFHMM hmm
```

For the rest of the POCTL* formulas two auxiliary functions are called, namely `extHMM` and `rename`, which also are implemented through pattern matching. The first one extends the original HMM recursively adding to the labels of states satisfying the most deeply state subformula $\Psi \notin AP$ its corresponding new atomic proposition a_{Ψ}. Ultimately, `extHMM` returns an extended HMM that has incorporated the new atomic propositions for every state subformula, including the original formula Φ itself. The second auxiliary function performs the substitutions of state subformulas Ψ by atomic propositions a_{Ψ} in a recursive fashion. At the end it returns the new atomic proposition a_{Φ} that replaces the original formula Φ itself. We can see that all the work is done by these two auxiliary functions. After invoking them, `satPOCTL` only calls itself once more with inputs the extended HMM and the final atomic proposition a_{Φ} as formula. For instance, to deal with the probabilistic operator `Prob c f path`, the function `satPOCTL` does

```
satPOCTL hmm (Prob c f path) = satPOCTL newHmm ap
    where
      (newHmm, _) = extHMM hmm (Prob c f path) 0
      (ap, _)     = rename (Prob c f path) 0
```

We perceive that both `extHMM` and `rename` functions take as the last argument an integer number, say `i`. The reason is that our atomic propositions are `String` values and we need to define new such elements, thus every time we create an atomic proposition we use this integer parameter. The brand new atomic proposition becomes `"ap"++show i`. Furthermore, the return value for these two functions is a pair where the second component is an integer; this is the next available number to define new atomic propositions. In the description of `extHMM` and `rename`, this second component is assigned to `i+1` after the atomic proposition `"ap"++show i` has been created and used.

To link this first stage with the next one, we have to highlight the situation when we encounter the probabilistic operator while trying to extend the given HMM. As it was said in Chapter 3, it is hard to find the states that satisfy this formula. Hence the subroutine `directApp` is called to obtain the states we are looking for. The way we invoke this subroutine and the details on how to extend the HMM in this case are illustrated with the next fragment of code; its clarification comes below.

```
extHMM hmm (Prob comp x path) i = (HMM {statesHMM = statesHMM hmm,
                                   pTransHMM = pTransHMM hmm,
                                   labelFHMM = newLabel,
                                   obsHMM = obsHMM hmm,
                                   pObsHMM = pObsHMM hmm,
                                   initDiHMM = initDiHMM hmm},
                                   j+1)
    where
      newLabel = listArray (bounds lab)
                       [if (elem (edos!k) sat)
                        then (lab!k)++["ap"++show j]
                        else (lab!k) | k<-indices lab]
      sat = directApp newHMM (fst (renamePath path i)) comp x
      lab = labelFHMM newHMM
      edos = statesHMM newHMM
      (newHMM, j) = extHMMPath hmm path i
```

Basically, the resulting extended HMM varies from the one as input only in the labelling function, which now appends to the current label the new atomic proposition for those states satisfying the probabilistic formula, i.e., the states returned by calling `directApp`. It is worth mentioning that to reach the deepest state subformulas analogous functions to `extHMM` and `rename` are required when examining path formulas, as it happens with the argument of the probabilistic operator. These functions are `extHMMPath`, which further extends the labels of the HMM states in the presence of a path formula, and `renamePath`, that replaces state subformulas by new atomic propositions inside the path formula. These two processes occur first and their results are passed to `directApp`, together with the comparison operator `comp` and the threshold value `x`. Finally, notice how the integer `i` is the last argument of `extHMMPath`, where it may be used to generate new atomic propositions. We know `j` is the subsequent value we can take to create the atomic proposition `"ap"++show j`, afterwards we increase this `j` by one.

4.4 Implementing Stage Two

The implementation of stage two makes up a new module named after the approach we use to deal with the QOS formula $(\phi, \bowtie p)$, that is, `DirectApproach.hs`. We resume where we finished last section, with the function `directApp`. Its inputs are: `hmm`, an HMM; `p`, a path formula; `comp`, a comparison operator; and `x`, a real valued threshold. The return value is the set of HMM states that satisfy the formula. As we can see below, `directApp` calls another function

```
directApp hmm p comp x =
                         satisfy hmm (satQOSHMM hmm) (transForm p) comp x
```

Where `satisfy` implements the comparison expressed in Corollary 3.9. We will address this function later on. At the moment we focus on the mappings `satQOSHMM` and `transform`, that carry out the tasks described in stage two of the model checking algorithm.

First, the DTMC $\mathcal{D}$ is constructed via `satQOSHMM`, which takes the source HMM `hmm` as input and transforms it according to the next code.

```
satQOSHMM hmm = DTMC {statesDTMC = getStates,
                      pTransDTMC = newTrans,
                      labelFDTMC = newLabel,
                      initDiDTMC = initialT}
  where
   tmpList    = [Base (s,o) | s<-elems (statesHMM hmm), o<-(obsHMM hmm)]
   tmpLength = length tmpList
   obsLen     = length (obsHMM hmm)
   getStates = listArray (1, tmpLength) tmpList
   newTrans   = array ((1,1), (tmpLength, tmpLength)) [((n,m),
                (pTransHMM hmm)!((div (n-1) obsLen)+1, (div (m-1) obsLen)+1)
                * (pObsHMM hmm)!((div (m-1) obsLen)+1, (mod (m-1) obsLen)+1))
                | n<-[1..tmpLength], m<-[1..tmpLength]]
   newLabel = array (1, tmpLength)
               [(i, (labelFHMM hmm)!((div (i-1) obsLen)+1))
               | i<-[1..tmpLength]]
   initialT = array (1, tmpLength)
               [(i, (initDiHMM hmm)!((div (i-1) obsLen)+1)
                * (pObsHMM hmm)!((div (i-1) obsLen)+1, (mod (i-1) obsLen)+1))
                | i<-[1..tmpLength]]
```

The new states are elegantly constructed using a HASKELL list comprehension that is formed as a Cartesian product of the HMM states and observations. We explain the data constructor `Base` in the next section.

Now, to compute the remaining items of the DTMC we see that the state index of interest can be referred to as `(div (i-1) obsLen)+1`, where `obsLen` is the cardinality of the set of observations, i.e., $|O|$, and `i` $\in$ `[1..tmpLength]`, with `tmpLength` representing the size of the new set of DTMC states, that is, $|S| \times |O|$. We wish `i` values $1, 2, \ldots, |O|$ to be assigned to state 1, `i` values $|O|+1, \ldots, 2 \cdot |O|$ to be assigned to state 2, and so on. Since `i` takes every number from 1 to $|S| \times |O|$, the desired state index is effectively found when adding 1 to the integer division of `i-1` by `obsLen`. Likewise, the expression `(mod (i-1) obsLen)+1` repeatedly calculates the observation indices $1, 2, \ldots, |O|$ for this transformation. Once the state and observation indices are known, the stage two operations that transform the HMM into a DTMC described in Section 3.2 can be carried out readily.

Remark 4.1. The lines of code depicted above show that the labelling function for the constructed DTMC $\mathcal{D}$ takes the corresponding labels of the HMM states. Nonetheless, according to stage two, the label of state (s, o) should be $L(s) \cup \{\Omega \subseteq \Theta \,|\, o \in \Omega\}$. Notice that the set $AP_{\mathcal{D}}$ is defined as the union of $AP_{\mathcal{H}}$ and $\{\Omega \,|\, \Omega \subseteq \Theta\}$. The latter set would imply to computing the power set of Θ, which is highly inconvenient and time consuming. Fortunately, to get around this difficulty we realise that the purpose of the label attached to each state is to know the atomic propositions that are true in it. We now have two kinds of LTL atomic propositions, the simple ones whose pattern is `Atom (Simple str)`, and the ones involving a set of observations, `xs`, formed through the pattern `Atom (ObSet xs)`. If we want to know whether a simple atomic proposition `Atom (Simple str)` holds in state (s, o), we just check if the string `str` is in this state label. In the case of checking an atomic proposition whose pattern is `Atom (ObSet xs)`, the state (s, o) satisfies it if $o \in$ `xs` because this means `xs` $\in \{\Omega \subseteq \Theta \,|\, o \in \Omega\}$. *Clearly, in this way we prevented the calculation of the power set of* Θ.

Second, the function `transform` takes a path formula and every time it encounters $\mathbf{X}_{\Omega}\varphi$ detaches the set of observations and joins it via a conjunction, i.e., it returns $\Omega \wedge \mathbf{X}\varphi$. All other operators are left intact. This leads to the code

```
transForm (FormP (AtomP s)) = Atom (Simple s)
transForm (FormP VerdadP)   = Verdad
transForm (FormP FalsoP)    = Falso
transForm (NoPath p)        = No (transForm p)
transForm (OPath p q)       = O (transForm p) (transForm q)
transForm (YPath p q)       = Y (transForm p) (transForm q)
transForm (NextPath xs p)   = Y (Atom (ObSet xs)) (Next (transForm p))
transForm (UntilBPath n p q)= UntilB n (transForm p) (transForm q)
transForm (UntilPath p q)   = Until (transForm p) (transForm q)
transForm _                 = error "This case shouldn't happen. Because
                              the formula we are considering is a maximal
                              state subformula"
```

The last case of the pattern matching is for the situation in which we have a state formula within a path formula ϕ, but this is impossible because we recursively found the most nested state formula to be $\mathcal{P}_{\bowtie p}(\phi)$.

We explain now the function `satisfy`, that finds, with the help of the induced DTMC `m`, the states of the HMM `hmm` that satisfy the LTL formula obtained by `transform`. For each state, we check that the probability of satisfaction is within the interval established by the comparison operator `comp` and the threshold `x`. Let us look at the way it works when the LTL formula is the next operator

```
satisfy hmm m (Next p) comp x = comparison hmm comp x
                                    (probVal myMC size (sat myMC ap))
  where
   hmmStat  = statesHMM hmm
   size     = rangeSize (bounds hmmStat)
   (myMC, _)= cxu m (Next p) 0
   (ap, _)  = reWrite (Next p) 0
```

Next, as dictated by the third stage of the model checking algorithm, the construction $C_{\mathbf{X}}$ is performed by calling the `cxu` function. The corresponding substitution is also carried out via the `reWrite` method. Their respective output is `myMC` and `ap`. Since new atomic propositions are constantly created by this third stage, we decided to take the same approach we used in stage one for the creation of such elements, i.e., new atomic propositions are brought about thanks to an extra integer argument, `i`, which increases by one after being used.

The relevant operation here is the summation found in Corollary 3.9. This quantity is computed by function `probVal` for every HMM state. Basically, it takes the DTMC `m` outcome of stage three, the number `size` of HMM states, and the set `sat` of DTMC states that satisfy the LTL formula (the process to determine the states satisfying an LTL formula appears in the next section). Then, for each state u in `sat`, a pair is produced with first component the original HMM state that eventually leads to u, and second component the initial distribution of u in `m`. Once these pairs are found, `probVal` matches the elements with same first component, i.e., same HMM state index, and sums their second components up. Before that, the respective summation for each HMM state had initial value of `0.0`. The argument `size` indicates how many initial value are required. So, the desired summation is calculated for all states in the HMM.

Lastly, the comparison of each value in `probVal` with the threshold number `x`, according to the operator `comp`, is done by the `comparison` function. If the individual comparison is evaluated to `True`, then the corresponding HMM state satisfies the probabilistic formula and is part of the output list. This, in turn, is the outcome of `satisfy`; consequently, it, too, is the outcome of `directApp`. If the comparison is `False`, the state does not satisfy the formula and is ignored.

4.5 Implementing Stage Three

The constructions that make stage three of the model checker are grouped in a module named `Courcoubetis.hs`, as one of the authors of [6], which describes them.

We begin this section by defining an adequate data type for the DTMC states that enables us to recursively create new states, as specified by the multiple constructions studied in stage three. We recall that these new states are now pairs (u, ξ_l), where ξ_l is an atomic proposition and u itself could be the outcome of a previous transformation, i.e., u might be a pair (u', ξ'_l). We proceed in this fashion until a basic state from the original DTMC is reached. Hence, the following recursive data type precisely captures the required specification

```
data Courcou a = Base a | Par (Courcou a, String)
```

From the code of the `satisfy` function given at the end of the last section, we found a call to `cxu`. This method implements the transformations $C_{\mathbf{X}}$, $C_{\mathcal{U}}$ and $C_{\mathcal{U}^{\leq n}}$. Directly linked with these constructions is the substitution process that replaces an innermost temporal operator inside an LTL formula by the atomic proposition ξ. The `reWrite` function implements this task. Both `cxu` and `reWrite` realise their work recursively. They also manage the creation and manipulation of atomic propositions with an index number, similar to the manner in which functions `extHMM` and `rename` work, as we saw earlier.

Before diving into the details of the `cxu` implementation, let us briefly discuss how we find the set of states that satisfy the propositional formula φ, denoted $\mathsf{Sat}(\varphi)$. This is achieved via the function `sat` whose inputs are the DTMC `m` and a propositional formula, for which it considers four base cases, i.e., `Verdad`, `Falso`, `Atom (Simple s)` and `Atom (ObSet xs)`. Noticeably, the computation of the set `Sat` for the last two formulas, as examined in Remark 4.1, has the encoding

```
sat m (Atom (Simple s)) = [edos!ind | ind<-indices labl, elem s (labl!ind)]
  where
   labl = labelFDTMC m
   edos = statesDTMC m
sat m (Atom (ObSet xs)) = [edos!ind | ind<-indices labl,
                                                 elem (getObs (edos!ind)) xs]
  where
   labl = labelFDTMC m
   edos = statesDTMC m
```

Where the function `getObs` receives a state of type `Courcou a`, and returns the observation `o` obtained by backtracking on this state up to `Base (s,o)`. According to the method `satQOSHMM`, `s` is an HMM state and `o` is in Θ.

The remaining situations are recursively evaluated by `sat`. Particularly, the set union is used to find the states that make a disjunction formula true. Likewise, the set intersection is computed to get the states satisfying a conjunction.

Next, we will explain how the `cxu` function operates in the case of the unbounded until operator. In HASKELL code we have

```
cxu m (Until phi1 phi2) n =
      (DTMC {statesDTMC = newEdos,
       pTransDTMC = newTransitionU newM newEdos sY sN sQ solVect,
       labelFDTMC = newLabellingU newM newEdos,
       initDiDTMC = newInitDistU newM newEdos sY sN solVect}
      , nextN)
  where
   (tmpM, tmpN)   = cxu m phi1 (n+1)
   (newM, nextN)  = cxu tmpM phi2 tmpN
   (p1, nn)       = reWrite phi1 (n+1)
   (p2, _)        = reWrite phi2 nn
   sN             = sNoU p1 p2 newM
   sY             = sYesU p1 p2 sN newM
   sQ             = sQMU newM sY sN
   solVect        = vectProbUU newM sY sQ
   newEdos        = newStatesU newM n sY sN sQ
```

Initially, the recursive call on the arguments of the until operator is made. We apply the function `cxu` to the current DTMC `m`, the formula `phi1`, and the next index number `n+1`. Its outputs become part of the input to the second application of `cxu`, but now the formula `phi2` is taken. In the same way, we call twice the function `reWrite`; first on `phi1`, afterwards on `phi2`. These initial procedures provide us with the actual DTMC `newM`, and formulas `p1` and `p2`, needed to carry out the current construction.

As described in stage three, the partitioning subsets S^{NO}, S^{YES} and $S^{?}$, that in the code above are `sN`, `sY` and `sQ`, respectively, have to be performed to continue with the other steps. Basically, we identify the states in each of the previous sets by means of the algorithms in Figures 3.2 and 3.3, e.g., $S^{\mathsf{NO}} = \textsc{ProbNo}(\mathsf{Sat}(\varphi_1), \mathsf{Sat}(\varphi_2))$, whose equivalent expression in HASKELL is

```
sNoU phi1 phi2 mC =
        (elems (statesDTMC mC)) \\ (probNo mC (map (getInd mC) satPhi1)
                                        satPhi2 (map (getInd mC) satPhi2))
  where
    satPhi1 = sat mC phi1
    satPhi2 = sat mC phi2
```

Note that the function `getInd` obtains the index of the passed state within the input DTMC `mC`. Furthermore, the formulas `phi1` and `phi2`, as well as the DTMC `mC`, are given to the function `sNoU`, which calculates a set subtraction as indicated by the last instruction of the PROBNO algorithm. The earlier instructions making up the rest of this algorithm are coded by the function `probNo` that appears next. Its parameters are: `mC`, a DTMC; `sat1Ind`, the indices of states satisfying `phi1`; `sat2`, the set of states satisfying `phi2`; and `myInd`, the indices of states satisfying `phi2`.

```
probNo mC sat1Ind sat2 myInd = if sat2 == nL
                               then sat2
                               else probNo mC (sat1Ind \\ newIndices) nL newIndices
  where
    nL = union sat2 [st!i | i<-sat1Ind,
                          findState (row i (pTransDTMC mC)) myInd]
    findState _ [] = False
    findState arr (n:ns) = if arr!n > 0
                                     then True
                                     else findState arr ns
    st = statesDTMC mC
    newIndices = (map (getInd mC) (nL \\ sat2))
```

Since Haskell lacks a loop expression, to implement the one involved in ProbNo we execute recursive calls that end up generating the same outcome. We have to point out that the `row` function takes a two-dimensional array and an index `i`. Its output is an one-dimensional array representing the `i` row of the input array. This function is taken from [26]. Notice also how the function `findState` determines whether or not there is a state s' in `sat2` with index `n`, such that the probability of going from the ith state s, that satisfies `phi1`, to s' is greater than zero. When the first such state s' is found, the `True` value is immediately returned by `findState`; otherwise, we proceed recursively. If no state meets these requirements, `findState` returns `False`. In this way the variable `nL` effectively finds the same values as R' in Figure 3.2. Importantly, in the function call `probNo mC (sat1Ind \\ newIndices) nL newIndices`, we remove from `sat1Ind` the elements of `newIndices`, i.e., the indices of states satisfying `phi1` just added to `nL`. So, in this function call we focus on any state left satisfying `phi1` that may reach states with untested indices in `newIndices`. The implementation of the sets S^{YES} and $S^{?}$ is analogous.

Next, the method `vectProbUU` computes the array `solVect` that holds the probabilities that $\varphi_1 \mathcal{U} \varphi_2$ is satisfied starting from every DTCM state. These magnitudes are the solution of a linear equation system (LES), with matrix form $\mathbf{M} \cdot \underline{q} = \underline{b}$, whose details are specified in stage three of the model checking algorithm. To obtain the solution of this LES we decided to import the library `Math.LinearEquationSolver`. In particular, inside `vectProbUU` the function `solverRationalLinearEqs` is invoked with the matrix $\mathbf{M}$, the vector $\underline{b}$ and the *Satisfiability Modulo Theories* solver *Z3* as input, see [4, 24]. The result is the solution vector $\underline{q}$ with the desired probabilities for every DTMC state.

At this point we are in a position to define the new DTMC built via the $C_{\mathcal{U}}$ construction. Initially, the function `newStatesU` computes the new set of states which has an intuitive implementation driven by its formal definition.

```
newStatesU mC n yes no qMark = array (1, length asoccL) (zip [1..] asoccL)
  where
   edos   = elems (statesDTMC mC)
   asoccL = [if elem s yes
```

```
             then Par (s,"xi"++show n)
             else if elem s no
                  then Par (s, "noXi"++show n)
                  else Par (s, "xi"++show n) | s<-edos]
             ++ [Par (s, "noXi"++show n) | s<-edos, elem s qMark]
```

The function parameters are the current DTMC `mC`, the index number `n`, and the partitioning subsets `yes`, `no` and `qMark`. Furthermore, `newstatesU` pairs up the DTMC states with a new atomic proposition, `"xi"++show n` or `"noXi"++show n` or both. The choice depends on which subset the state belongs to. Now, it is clear the usage the index number `n` has when creating new atomic propositions.

On these same lines, the new labelling function defines the label of state (u, ξ_l) in `newStat` just by adding the second component, ξ_l, to u's label. This is exactly stated by the `newLabellingU` function.

```
newLabellingU mC newStat = listArray (bounds newStat)
                           [getAtom edo | edo<-elems newStat]
  where
   getAtom s = (labelFDTMC mC)!(getInd mC (first s)) ++ [second s]
```

The functions `first` and `second` return the first and second component of a value of type `(Courcou a)`, respectively, as long as this value follows the pattern `Pair (Coucou a, String)`.

We concentrate next on the function `newInitDistU` that finds the initial distribution for the new set of states.

```
newInitDistU mC newStat yes no vect =
         listArray (bounds newStat)
         [computeInitDis (getInd mC (first edo)) edo | edo<- elems newStat]
  where
   computeInitDis indU (Par (u, atom))
            | (elem u yes) || (elem u no)=  initDist!indU
            | (take 2 atom /= "no")      = (initDist!indU)*(vect!indU)
            | otherwise                  = (initDist!indU)*(1-(vect!indU))
   computeInitDis _ _ = error "This case shouldn't happen"
   initDist = initDiDTMC mC
```

For each new state `Par (u, atom)` in `newStat`, we check whether `u` is a member of either the partitioning subsets `yes` or `no`. If so, its respective initial distribution is the same as the one of `u` in the DTMC `mC`. If that is not the case, i.e., `u` is in $S^?$, we look at the atomic proposition `atom`. If `atom` $\neq \neg\xi$ (equivalently, `atom` $= \xi$), then the $C_\mathcal{U}$ construction says that the initial distribution is $\pi_u^{\texttt{mC}} \cdot q_u$, that is in code, `(initDist!indU)*(vect!indU)`, where the vector `vect` already holds the values q_u for every state u in `mC`. If `atom` $= \neg\xi$, we know that the initial distribution turns out to be $\pi_u^{\texttt{mC}} \cdot \overline{q}_u$, that is, `(initDist!indU)*(1-(vect!indU))`. Notably, since we are computing the initial distribution of new states that necessarily have the form `Par (u, atom)`, it would be an error to do it for the other plausible value `Base a`.

The new transition probability matrix is the only element left to define. It is attained by the function `newTransition` that receives the source DTMC `mC`, the set `newStat` of new states, the three partitioning subsets (`yes`, `no` and `qMark`), and the vector `vect` with the probabilities q_u for every DTMC state u.

```
newTransitionU mC newStat yes no qMark vect = array ((1,1), (sb, sb)) lista
  where
   sb     = snd (bounds newStat)
   lista  = [ ((i,j), getValU mC (newStat!i) (newStat!j) yes no qMark vect)
            | i<-[1..sb], j<-[1..sb]]
```

The probability of going from state `Par (u,x)` to state `Par (v,y)` is the result of the auxiliary function `getValU`, which is shown below.

```
getValU mC (Par (u,x)) (Par (v,y)) yes no qMark vect
  | (elem u yes || elem u no)= if (elem v yes || elem v no)
                               then pTran!(gu, gv)
                               else if (take 2 y /= "no")
                                    then (pTran!(gu, gv))*(gQv)
                                    else (pTran!(gu, gv))*(1-gQv)
  | elem v qMark             = if (take 2 x /= "no")
                               then if (take 2 y /= "no")
                                    then (pTran!(gu, gv))*(gQv)/(gQu)
                                    else 0
                               else if (take 2 y == "no")
                                    then (pTran!(gu, gv))*(1-gQv)/(1-gQu)
                                    else 0
  | elem v yes               = if (take 2 x /= "no")
                               then (pTran!(gu, gv))/(gQu)
                               else 0
  | elem v no                = if (take 2 x == "no")
                               then (pTran!(gu, gv))/(1-gQu)
                               else 0
  | otherwise                = 0
  where
   gu = getInd mC u
   gv = getInd mC v
   gQu = vect!gu
   gQv = vect!gv
   pTran = pTransDTMC mC
```

Expectedly, the step of stage three depicting all the cases needed to obtain the transition probabilities is directly mapped into the previous code. As a toy example, let us look at the situation when we seek the probability value of going from `Par (u,"xi1")` to `Par (v,"xi1")`, where `u` $\in S^?$ and `v` $\in S^?$. Moreover, `(take 2 x /= "no")` and `(take 2 y /= "no")` are true because both `x` and `y` represent the atomic proposition `"xi1"`. Therefore the return value is `(pTran!(gu, gv))*(gQv)/(gQu)`, i.e., $a_{u\,v}{*}q_v/q_u$, as it is established by stage three of the model checking algorithm.

The construction $C_{\mathcal{U}}$ is completed now. All four DTMC elements have been already defined in Haskell, namely the set of states, the transition probabilities, the labelling function and the initial distribution. Furthermore, the remaining construction encodings for $C_{\mathbf{X}}$ and $C_{\mathcal{U}^{\leq n}}$ are much like the one we studied for $C_{\mathcal{U}}$.

Regarding the way this model checker works we must say that, in addition to the code explained in the preceding pages, a basic lexer and parser were implemented, as well as a main module named `Main.hs` that sequentially asks for inputs and calls the appropriate methods to get the model checker executed. The inputs given to the main module are an HMM and a POCTL* formula; the former is passed as a file with extension `.poctl` and the latter is directly typed into the terminal. In order to gain insight on how the `.poctl` files are arranged, Appendix D exhibits a sample input file containing an HMM. It also illustrates the way we pass an input POCTL* formula to our model checker.

This chapter gave the implementation of key aspects of the model checker for POCTL*, such that the rest of the code can be easily extended from the ideas presented here. This encoding was written in Haskell, programming language that allowed us to come up with elegant, simple and straightforward translations from the formal definition presented in Chapter 3 to the code shown here. Furthermore, we provided solutions to overcome difficulties such as the apparent need to compute the power set of the observation set and the lack of a loop expression. Importantly, a real-world application is studied in the next chapter. There a handover task from a robot to a human is modelled through an HMM, which is considered by the POCTL* model checker to verify relevant properties of this human-robot interaction. Also, an interesting technique is considered to reduce the number of observations, thus the model checking process performs in reasonable running time.

Chapter 5

Verification of a Human-Robot Interaction

Human-assistive robots are machines designed to improve our quality of life by helping us to achieve tasks [11, 12]. Moreover, some situations may require robots to be powerful and therefore potentially dangerous. As a result, trustworthiness must be guaranteed to actually let the robot interact with the general public. Especial attention is paid on properties of usefulness and safety, which take the form of specifications. Intuitively, on the one hand, a usefulness or liveness property stipulates that a *good thing* happens during the execution of the system [1]. On the other hand, a safety property stipulates that some *bad thing* does not happen during execution [1]. Consequently, to demonstrate the correctness and trustworthiness of the robot's behaviour we must prove that the model of the robot control system satisfies a set of liveness and safety properties. This verification process is performed by a model checker [11].

For the sake of working out a real-world example of the verification process carried out by the model checker for POCTL* coded in Chapter 4, this chapter studies the robot-to-human object handover task [11, 12]. The robot's decision to release the object is determined by a Hidden Markov Model that identifies the different stages occurring throughout the handover by its states. To construct the desired HMM, initial estimated values for each probability matrix are extracted out of a sequence of experiments. For example, the initial observation values are obtained from the robot's fingers while executing several complete handover tasks. Then a training algorithm is applied on the initial estimated values to better approximate the probability quantities involved in the model. To release the object, the system is required to reach a state in which it is safe to do so. Since an HMM is the way this interaction is described, it makes sense to use our model checker for POCTL* to know whether some liveness and safety properties hold under this interaction.

The platform used to study this human-robot interaction (HRI) was Bert2 [9, 10, 20]. We start by describing Bert2's infrastructure. Then, the construction of the HMM modelling the handover task is explained. Here we introduce the concept of *vector quantisation*, a technique used to bring down the number of observations by partitioning the input vector space into few regions. Furthermore, the two training processes mentioned in Chapter 1 as the solution of Problem 3 of HMMs are considered to adjust the probability matrices of the HMM being constructed. Finally, some specifications, i.e., liveness and safety properties, are proposed in order to guarantee the correctness and trustworthiness of this interaction. Our model checker takes these specifications and the trained HMM as inputs, and returns the states satisfying the corresponding property. The results are interpreted with respect to the confidence expected from this HRI interaction.

5.1 Bert2 Infrastructure

Bert2 (acronym for Bristol Elumotion Robot Torso 2) [9, 10, 20] is a humanoid robot designed at the Bristol Robotic Laboratory (BRL) in co-operation with Elumotion, a mechanical engineering company. It is composed of a torso, shoulders, arms and a head. These components contain multiple joints that can be independently controlled. Each arm has an anthropomorphic hand with multiple degrees of freedom that implements higher-level movement commands, e.g., we can instruct it to *prepare to grasp*, *grasp* and *release an object*. Remarkably, these actions are done by Bert2 in a human-like fashion, which allows an improved study of the object handover scenario (see Figure 5.1).

Furthermore, the joints present in Bert2 are controlled through EPOS motor controllers which are connected to a *controller area network* (CAN) that, together with a central-controller, that is, a Linux computer with a PCI to CAN interface, makes up the joint-level control infrastructure. The central-controller is in charge of broadcasting the synchronisation and heartbeat messages. The EPOS units expect this heartbeat messages at a regular basis of 20ms to broadcast their internal state. This guarantees that there is always a link between the EPOS units and the central-controller. The Bert2's functionality described so far is enough to cover the basic handover scenario we are interested, which gives a real-world application of the POCTL* model checker. Nonetheless, we will briefly mention the remaining elements available in Bert2's platform.

The purpose of the design of Bert2's head is to implement the facial expression communication channel as used in human-human interaction, with a particular emphasis on gaze but keeping an artificial, "robotic" look. This plastic head incorporates a colour LCD screen that allows Bert2 to displaying eyes, eyebrows, mouth and lips. Thus a wide rage of face expressions are possible, and changing between them is seamlessly.

Bert2 also features a gaze tracking system from Seeing Machines called faceLAB^{TM}. This includes two cameras mounted on either side of Bert2's head that track the head and eyes of a human directly in front of it. The cameras use an infrared stereo vision system to detect head pose and position. Additionally, cornea reflection techniques achieve high-accuracy eye tracking.

The Bert2's framework integrates the CSLU Toolkit Rapid Application Development in order to interact with the user through spoken language. Moreover, the construction of speech dialogues is supported via a state-based graphical programming environment.

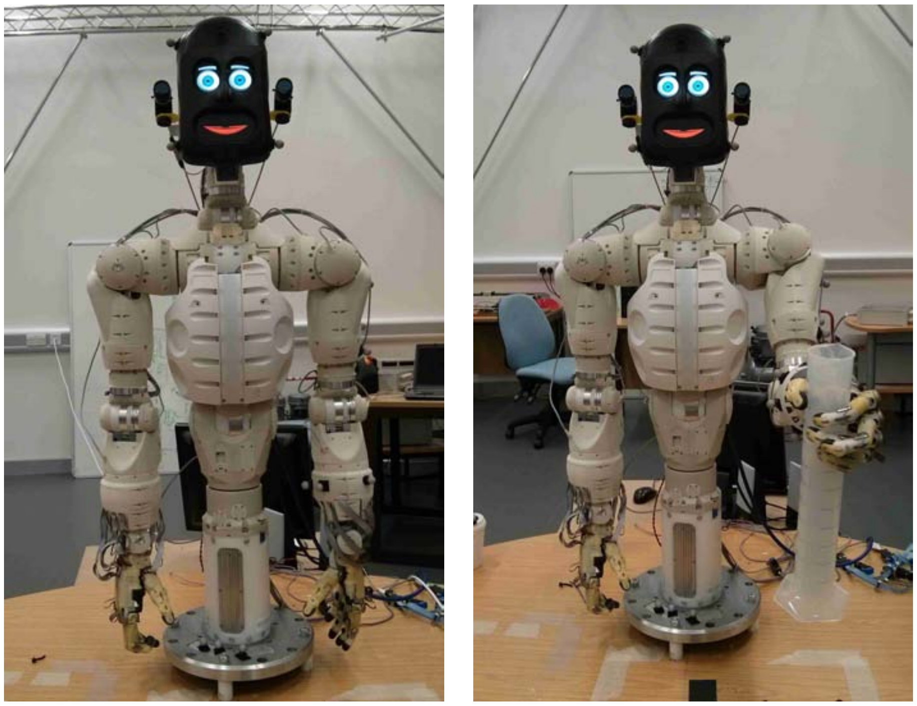

(a) Bert2 in his initial position. **(b)** Bert2 handing an object over to a person.

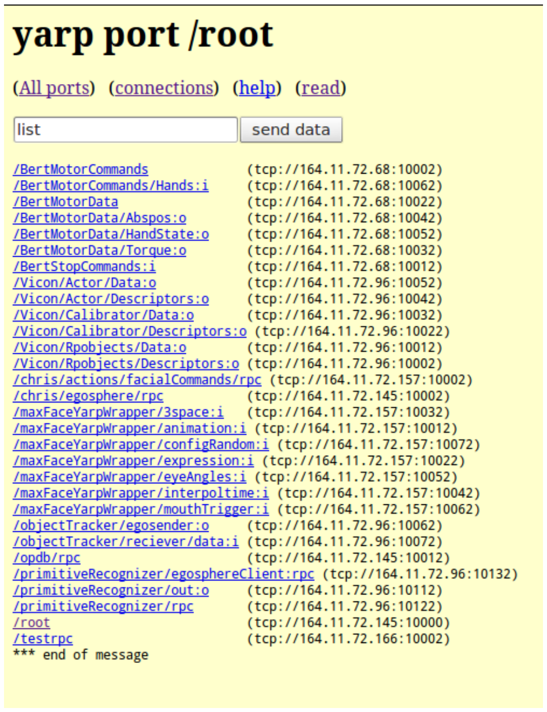
Figure 5.1: Bert2 platform.

The platform can track objects and human body parts in 3D space, positioned in the same room that Bert2, via the VICON motion capture system. Every item to be tracked must have attached at least three retro-reflective markers. Furthermore, the VICON system represents these items with two databases handling static and dynamic knowledge. The *OPDB* (Object Property Data Base) is the common namespace manager and stores the static components of all objects in the interaction scenario. The *EgoSphere* module quickly, dynamically and asynchronously stores object positions and orientations, matches them with the *OPDB* object ID, and broadcasts this on YARP ports.

The robotic system's communication is based on YARP (Yet Another Robotic Platform), an open-source C++ project that reduces the complications of infrastructure-level software development by producing independent software modules that connect with each other using named TCP/IP or UDP ports. Figure 5.2 shows a list of several YARP ports running on Bert2, e.g, `/BertMotorCommands/Hands:i`. These channels can be considered as a data streaming interface or as remote-procedure-call facilities.

5.2 HMM implementation

We break down the states of the HMM corresponding to the basic handover interaction as proposed in [10, 12], where four states are chosen, namely



```
/BertMotorCommands              (tcp://164.11.72.68:10002)
/BertMotorCommands/Hands:i      (tcp://164.11.72.68:10062)
/BertMotorData                  (tcp://164.11.72.68:10022)
/BertMotorData/Abspos:o         (tcp://164.11.72.68:10042)
/BertMotorData/HandState:o      (tcp://164.11.72.68:10052)
/BertMotorData/Torque:o         (tcp://164.11.72.68:10032)
/BertStopCommands:i             (tcp://164.11.72.68:10012)
/Vicon/Actor/Data:o             (tcp://164.11.72.96:10052)
/Vicon/Actor/Descriptors:o      (tcp://164.11.72.96:10042)
/Vicon/Calibrator/Data:o        (tcp://164.11.72.96:10032)
/Vicon/Calibrator/Descriptors:o (tcp://164.11.72.96:10022)
/Vicon/Rpobjects/Data:o         (tcp://164.11.72.96:10012)
/Vicon/Rpobjects/Descriptors:o (tcp://164.11.72.96:10002)
/chris/actions/facialCommands/rpc (tcp://164.11.72.157:10002)
/chris/egosphere/rpc            (tcp://164.11.72.145:10002)
/maxFaceYarpWrapper/3space:i    (tcp://164.11.72.157:10032)
/maxFaceYarpWrapper/animation:i (tcp://164.11.72.157:10012)
/maxFaceYarpWrapper/configRandom:i (tcp://164.11.72.157:10072)
/maxFaceYarpWrapper/expression:i (tcp://164.11.72.157:10022)
/maxFaceYarpWrapper/eyeAngles:i (tcp://164.11.72.157:10052)
/maxFaceYarpWrapper/interpoltime:i (tcp://164.11.72.157:10042)
/maxFaceYarpWrapper/mouthTrigger:i (tcp://164.11.72.157:10062)
/objectTracker/egosender:o      (tcp://164.11.72.96:10062)
/objectTracker/reciever/data:i (tcp://164.11.72.96:10072)
/opdb/rpc                       (tcp://164.11.72.145:10012)
/primitiveRecognizer/egosphereClient:rpc (tcp://164.11.72.96:10132)
/primitiveRecognizer/out:o      (tcp://164.11.72.96:10112)
/primitiveRecognizer/rpc        (tcp://164.11.72.96:10122)
/root                           (tcp://164.11.72.145:10000)
/testrpc                        (tcp://164.11.72.166:10002)
*** end of message
```

Figure 5.2: The YARP port list.

1. The robot is not holding the cup.
2. The robot is grabbing the cup.
3. The robot is holding the cup (with the user not touching it).
4. The user is grabbing the cup.

Note that our initial state is the last one of the model given in [10, 12]. The state machine of this HMM, without observations, is depicted in Figure 5.3. For convenience we start counting the states from 1.

The initialisation of the model's matrices A, B and π is the first step to discover the final HMM that represents the handover task, we address this assignment next.

The process starts at state Robot not hold, so its initial distribution value π_1 is almost one, whereas the other states have been given initial distribution values close to zero.

Moreover, the transition probability matrix A must encourage the transitions appearing in Figure 5.3. We see that for each state there are two outgoing edges. Thus for each row of A the cells corresponding to the two outgoing edges have approximate values of 0.5. The remaining cells in the row hold magnitudes close to zero.

To initialise the observation probability matrix B, we have to choose the source of input observations. Taking advantage of the highly specialised anthropomorphic hands, we focus on the values returned by them during the handover interaction. Their output is read from the port `/BertMotorData/HandState:o` and has the format:

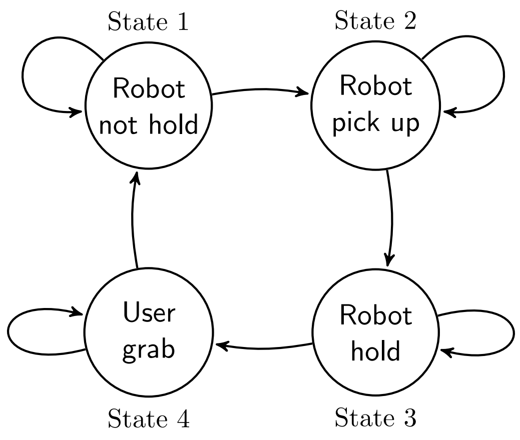


Figure 5.3: The basic HMM for the handover process.

```
[Time stamp] [Left hand field set] [Right hand field set].
```

Each field set has the form:

```
[Hand live] [Hand state] [Hand finished]
     [Hand palm pressure] [c1] [c2] [c3] [c4] [c5] [c6] [c7] [c8].
```

where `Hand live` is a positive number or 0, the latter means the hand is not active, otherwise the hand is working properly; `Hand state` is a value taken from the set $\{-1, 0, 1, 2, 3, 4, 5, 6, 7\}$ that indicates the instruction being executed, e.g., *prepare to grasp* or *release*; `Hand finished` is 0 if the task is not completed, it is 1 if the task is finished; `Hand palm pressure` is an integer value returned by the hand touch sensor; `c1-c8` are motor currents[9] of particular hand joints according to the following table.

c1	$\rightarrow$	index finger *metacarpophalangeal joint* (MCPJ)
c2	$\rightarrow$	middle finger *proximal interphalangeal joint* (PIPJ)
c3	$\rightarrow$	index finger PIPJ
c4	$\rightarrow$	middle finger MCPJ
c5	$\rightarrow$	ring finger gross flexion
c6	$\rightarrow$	small finger gross flexion
c7	$\rightarrow$	thumb opposition
c8	$\rightarrow$	thumb flexion

Since BERT2's right hand was not functioning at the moment we were working at the BRL, we completely ignore its values. Thus BERT2's left hand is used to accomplish the handover process.

To understand how the left hand finger values change throughout the interaction, we carried out fifty basic handover tasks on BERT2, in which the robot simply takes an object from a table and then hands it over to a person. So we could identify the precise

[9]The motor currents are provided in a fraction (i.e., $\frac{1}{8}$) of the actual motor current in mA.

instant the scenario moves from one state to the next. For each trial we were able to save in a text file the output values. We later examined this data to determine the most distinctive and meaningful values, among the `c1-c8` motor currents, that exhibit important variations every time a state transition occurs. Under BERT2's particular settings two current values are vital to perform this handover task, namely the index and middle finger MCPJ motor current values.

Therefore, we decide to consider as observations the ordered pairs whose first and second components are the index and middle finger MCPJ current values, respectively, i.e., the observations have the form (`c1`, `c4`). Nevertheless, when trying to find the number of different values these motor currents might take, we realise that there are nearly 236 possibilities for the index finger MCPJ and 239 options for the middle finger MCPJ. Hence the number of ordered pairs, i.e., observations, is up to 56404. This exceeds the capabilities of our model checker, then a way to bring down the number of observation is necessary. Fortunately, we can achieve this goal by means of the vector quantisation technique.

5.2.1 Vector Quantisation

An N-level k-dimensional vector quantiser [21, 23], also called block quantiser, is a mapping, q, that assigns to each input vector, $\mathbf{x} = (x_0, \ldots, x_{k-1})$, a codeword, $\mathbf{y} = q(\mathbf{x})$, drawn from a finite codebook, $\mathbf{Y} = \{\mathbf{y_i} \,|\, \mathbf{i} = 1, \ldots, N\}$. The vector quantiser q is described by the codebook $\mathbf{Y}$ and the partition, $C = \{C_1, \ldots, C_N\}$, of the input vector space into the sets $C_\mathbf{i} = \{\mathbf{x} \,|\, q(\mathbf{x}) = \mathbf{y_i}\}$ of input vectors $\mathbf{x}$ associated by q to the $\mathbf{i}$th codeword $\mathbf{y_i}$, we call the sets $C_\mathbf{i}$ cells.

A distortion measure obtains the error produced by considering the codeword $\mathbf{y}$ instead of the actual input vector $\mathbf{x}$, this quantity must be non-negative and is denoted by $d(\mathbf{x}, \mathbf{y})$. Many distortion measures have been proposed in the literature. However, we will use the most common and mathematical convenient, i.e., the squared-error distortion that is computed as follows:

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=0}^{k-1} |x_i - y_i|^2.$$

To design a N-level (or N-size) codebook, we have to partition the k-dimensional space into N cells $C_1, \ldots, C_N$ each one connected with its matching codeword $\mathbf{y_i}$. In order to bring on an optimal minimum-distortion quantiser two conditions must be met. First, the optimal quantiser is realised by using a minimum-distortion or nearest neighbour selection rule, i.e., for every input vector $\mathbf{x}$ we have

$$q(\mathbf{x}) = \mathbf{y_i} \quad \text{iff} \quad d(\mathbf{x}, \mathbf{y_i}) \leq d(\mathbf{x}, \mathbf{y_j}), \quad \mathbf{j} \neq \mathbf{i}, \quad 1 \leq \mathbf{j} \leq N.$$

In other words, the quantiser chooses the codeword which distortion with respect to $\mathbf{x}$ is minimum. If a tie happens, then the codeword with lowest index is taken as the output of the quantiser.

The second required condition for optimality is that each codeword $\mathbf{y_i}$ is chosen to minimise the average distortion $D_\mathbf{i}$ in cell $C_\mathbf{i}$. Since we are provided with a set of training vectors $\{\mathbf{x_j} \,|\, \mathbf{j} = 0, \ldots, n-1\}$, the expression for $D_\mathbf{i}$ is

$$D_\mathbf{i} = \frac{1}{N_\mathbf{i}} \sum_{\mathbf{x_j} \in C_\mathbf{i}} d(\mathbf{x_j}, \mathbf{y_i}),$$

where $N_\mathbf{i}$ is the number of training vectors in the cell $C_\mathbf{i}$. For the squared-error criterion, $D_\mathbf{i}$ is minimised by the Euclidean centre or gravity centroid, i.e.,

$$\mathbf{y_i} = \mathsf{cent}(C_\mathbf{i}) = \frac{1}{N_\mathbf{i}} \sum_{\mathbf{x_j} \in C_\mathbf{i}} \mathbf{x_j},$$

So, the next algorithm describes how to find a vector quantiser that minimises the average distortion for a training vector sequence.

Let N be the number of levels, $\epsilon \geq 0$ be the distortion threshold, $\mathbf{Y}_0$ be an initial N-level codebook and $\{\mathbf{x_j} \,|\, \mathbf{j} = 0, \ldots, n-1\}$ be a training sequence.

1. Set the iteration index $\mathbf{m} = 0$ and the average distortion $D_{-\mathbf{1}} = \infty$.

2. Given the codebook $\mathbf{Y_m} = \{\mathbf{y_i^m} \,|\, \mathbf{i} = 1, \ldots, N\}$, classify the training vectors into sets of the minimum distortion partition $\mathbf{P}(\mathbf{Y_m}) = \{C_1^\mathbf{m}, \ldots, C_N^\mathbf{m}\}$, such that $\mathbf{x_j} \in C_\mathbf{i}^\mathbf{m}$ if $d(\mathbf{x_j}, \mathbf{y_i^m}) \leq d(\mathbf{x_j}, \mathbf{y}_{\boldsymbol{\ell}}^\mathbf{m})$, for all $1 \leq \boldsymbol{\ell} \leq N, \boldsymbol{\ell} \neq \mathbf{i}$.

3. Compute the overall average distortion

$$D_\mathbf{m} = D(\{\mathbf{Y_m}, \mathbf{P}(\mathbf{Y_m})\}) = \frac{1}{n} \sum_{\mathbf{j}=0}^{n-1} \min_{\mathbf{y} \in \mathbf{Y_m}} d(\mathbf{x_j}, \mathbf{y}).$$

   If $(D_{\mathbf{m}-1} - D_\mathbf{m})/D_\mathbf{m} \leq \epsilon$, stop and return $\mathbf{Y_m}$ as the final codebook. Otherwise continue.

4. Update the codebook taking the centroids for each set in $\mathbf{P}(\mathbf{Y_m})$. Thus we obtain $Y_{\mathbf{m}+1} = \{\mathbf{y_i^{m+1}} \,|\, \mathbf{i} = 1, \ldots, N\}$, where $\mathbf{y_i^{m+1}} = \mathsf{cent}(C_\mathbf{i}^\mathbf{m})$. Set $\mathbf{m} = \mathbf{m} + 1$ and go to step 2.

From the four previous steps we can notice and be concerned that only partitions of the training sequence are considered. Nonetheless, once the algorithm terminates and the final codebook $\mathbf{Y_m}$ is found, it is used on new data, not pertaining to the training sequence, with the above mentioned optimum nearest neighbour rule, i.e, the codebook $\mathbf{Y_m}$ defines an optimum partition of the k-dimensional Euclidean space.

For our purposes, we are interested only on the cell $C_\mathbf{i}^\mathbf{m}$ where the input vector $\mathbf{x}$ lies after applying the quantiser q, thus the answer given by q when processing $\mathbf{x}$ is the index $\mathbf{i}$. In other words, our final observations are no longer ordered pairs but cells indices.

Step 1 of the vector quantisation algorithm requires an initial codebook $\mathbf{Y_0}$. In [21] two approaches are given to attain this initial codebook. We regard the first of them which uses a uniform quantiser over all or most of the training sequence if it is bounded, as is the case of the motor current values we get from BERT2 during the handover interaction.

Since our input vectors are pairs of motor current values, the choice we made for the initial codebook $\mathbf{Y_0}$ was to use a 2-dimensional uniform quantiser on a rectangle including all or most of the points in the training sequence. Specifically, the rectangle we used has vertices $(-124.872, -118.538)$, $(107.718, -118.538)$, $(-124.872, 119.487)$ and $(107.718, 119.487)$[10]. Furthermore, we define on this rectangle's edges codewords every 30 units. Hence, the initial codebook has length 32 which is still large to be taken as the number of possible observations. However, we decide to execute the vector quantiser algorithm with this initial $\mathbf{Y_0}$, distortion threshold $\epsilon = 0.01$ and training sequence the pairs of output values (`c1`, `c4`) from one of the experiments we performed on BERT2 that is considered as representative.

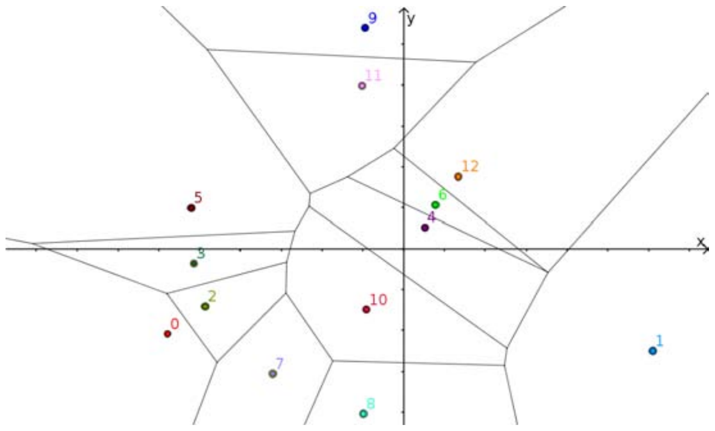
The idea is to perform step 2 in a way that if we get an empty cell $C_{\mathbf{i}}^{\mathbf{m}}$, then we drop its corresponding codeword $\mathbf{y_i^m}$ from the codebook. The resulting quantiser is optimum since the two conditions for optimality are held. We ended up having a codebook containing 13 cell indices, i.e., 13 regions into which the 2-dimensional space is partitioned as illustrated in Figure 5.4. So the observation set becomes $\Theta = \{0, 1, 2, \ldots, 12\}$. This small number of observation makes our model checker run in a practical amount of time. Finally, we spot from Figure 5.4 that the cells or regions produced by the vector quantisation algorithm defined a Voronoi diagram. As stated by [25], given a set of two or more but a finite number of distinct points in the Euclidean plane, a *planar ordinary Voronoi diagram* is the result of associating all locations in that space with the closest member(s) of the point set with respect to the Euclidean distance.

From each of the fifty experiments completed on BERT2 we draw an input observation sequence of pairs (`c1`, `c4`) on which we applied the previously described quantiser. We use these results to initialise the HMM matrix B. The entry $b_j(m)$ in B is the result of dividing the number of times observation (i.e., cell index) m occurred in state j by the total number of observations occurring in this particular state.

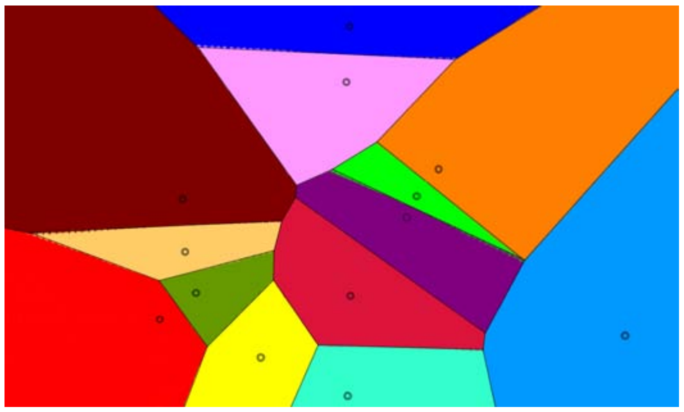
Once the initial estimates of the matrices π, A and B are computed, we adjust these matrices by executing either the Baum-Welch or the K-means algorithm. Both methods were studied in Chapter 1.

The implementation of the vector quantisation process was done in the JAVA programming language. In this same language, the Baum-Welch and K-means reestimation algorithms were coded. We compare the outcomes of both methods to gain confidence in their implementation. Furthermore, to successfully perform the multiplication operations required by the Baum-Welch algorithm we used the package *Apfloat*, a high performance arbitrary precision arithmetic library.

[10]The average of the lowest and highest values for the field `c1` considering all fifty experiments is -124.872 and 107.718, respectively. Similarly, the average of the lowest and highest values for `c4` is -118.538 and 119.487, respectively.



(a) The region boundaries and their positions with respect to the x and y axis.



(b) The regions filled with different colours. This partition can be seen as a Voronoi diagram.

Figure 5.4: The 2-dimensional space partitioned into 13 regions.

5.3 Verifying Liveness and Safety Properties

There is one HMM element that remains to be defined. It is the labelling function L that returns the set of atomic propositions satisfied by each state. Figure 5.5 shows the state machine for the basic handover task, next to each state the set of its valid atomic propositions is defined. We are considering the set of atomic propositions $AP = \{\texttt{rnh}, \texttt{rpu}, \texttt{rh}, \texttt{hg}\}$. Now the HMM is complete and can be formatted according to the instructions found in Appendix D to create the `handover.poctl` file that is passed to our model checker. Since we want to know whether each state satisfies the specification in turn, despite of how likely the state is initially chosen, the approach we follow is the proposed by Zhang in [35]. Hence the model checker is used as specified by Remark D.1.

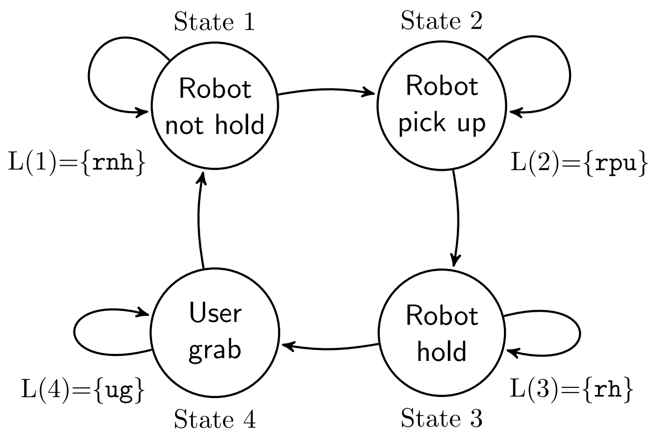


Figure 5.5: The labelled states involved in the basic handover process.

5.3.1 Liveness properties

A liveness property indicates that a *good thing* does happen during the execution of the system or program. In a probabilistic context, a liveness property is a claim that a *good thing* happens with high probability. Now we list some good things we expect to take place while executing the handover interaction with Bert2.

- *With probability at least 0.9,* Bert2 *releases the object when the user grabs it.* Clearly, we take the probabilistic operator to specify that the property holds with a probability of at least 0.9. If Bert2 is going to release an object, then it is holding it and does it until the user grabs the object, who waits for Bert2 to release it. Consequently, the POCTL* formula for this property is

$$\mathcal{P}_{\geq 0.9}(\texttt{rh} \wedge \texttt{rh}\,\mathcal{U}(\texttt{ug} \wedge \texttt{ug}\,\mathcal{U}\,\texttt{rnh})).$$

  When we pass it to our model checker together with the `handover.poctl` file, the set of states that is returned consists only of state 3, i.e., Robot hold. This means that with high probability Bert2 releases the object when the user grabs it, assuming Bert2 is holding it at the starting point.

- Recall that we are extracting the pairs (`c1`, `c4`) from the motor current values of Bert2's left hand finger joints. Thus an observation in our HMM is the region index the pair (`c1`, `c4`) is associated with by the vector quantiser q. Interestingly, these regions can be thought of as specific finger positions described by the pairs of finger motor values within the region.

  In Figure 5.6, we have plotted the observations obtained by a handover execution as a function of time. From this plot we would like the next property to hold. *The model generates the sequence of observations, i.e., the sequence of finger positions,* $\mathcal{O} = \{o_0, o_1, o_2, o_3\}$ *where* $o_0, o_1 \in \{2, 3, 5\}$ *and* $o_2, o_3 \in \{2, 3, 10\}$*, with* *a probability greater than 0.88.* This property becomes the POCTL* formula

$$\mathcal{P}_{>0.88}(\mathbf{X}_{\{2,3,5\}}(\mathbf{X}_{\{2,3,5\}}(\mathbf{X}_{\{2,3,10\}}(\mathbf{X}_{\{2,3,10\}}\,\mathsf{true})))).$$

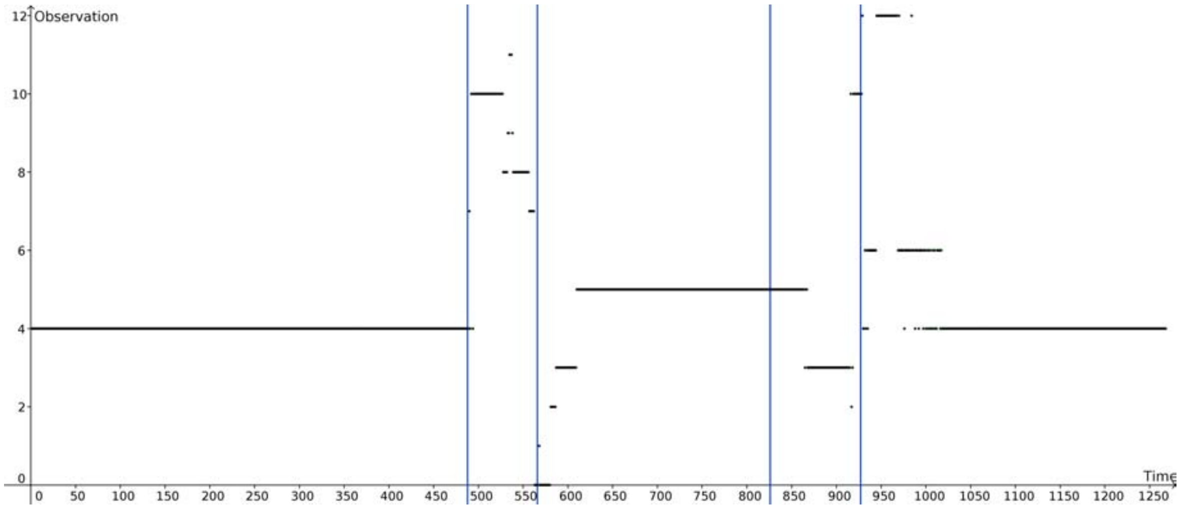


Figure 5.6: The plot of observations *vs* time. The first blue vertical line denotes the transition from state 1 to state 2. The second one indicates the transition from state 2 to state 3. And so on.

The output returned by the model checker is state 4 only. Furthermore, in HASKELL we can use the function `trace`, belonging to the `Debug.Trace` library, to print intermediate results. By means of the function `trace` we realise that the probability of satisfaction of the previous specification for states 1, 2, 3 and 4 is 4.998e-10, 4.086e-6, 7.509e-3 and 0.891, respectively. Noticeably, this formula is an extension of Rabiner's Problem 1 (see Section 1.2.1), since we have found the probability of the sequence of observations $\mathcal{O}$, such that at each instance the observation is drawn from a set of possible values.

- Analysing Figure 5.6 and other outcomes of the handover executions on BERT2, we identify all observations that state 3 (Robot hold) produces as $\{0,1,2,3,5,7\}$. Likewise, state 4 (User grab) mainly generates observations $\{2,3\}$. We wish to show that the next property holds. *With probability greater than 0.9 the user will grab the object resulting in* BERT2*'s finger positions 2 or 3, such that for every previous moment the object was held by* BERT2 *having any of the following finger positions* $\{0,1,2,3,5,7\}$. The corresponding POCTL* formula is

$$\mathcal{P}_{>0.9}((\mathsf{rh} \wedge \mathbf{X}_{\{0,1,2,3,5,7\}}\mathsf{true})\ \mathcal{U}\ (\mathsf{ug} \wedge \mathbf{X}_{\{2,3\}}\mathsf{true})).$$

we denote it as Φ.

When our model checker takes Φ together with the `handover.poctl` file, it returns states 3 and 4 with probabilities of satisfaction 0.9854 and 0.9855, respectively. These probability values were detected by calling the function `trace`.

Notice that observation 5 is the most important in state 3 as seen in Figure 5.6. If we drop observation 5 from Φ, the probability of satisfaction of state 3 dramatically falls to 1.829e-3. Note that if we keep this observation and drop any other (i.e., 0, 1, 2, 3 or 7) from Φ, its probability of satisfaction is greater at least by one order of magnitude than the one obtained when observation 5 was dropped. With such importance this observation is regarded that state 3 satisfies $(\mathsf{rh} \wedge \mathbf{X}_{\{5\}}\mathsf{true})\,\mathcal{U}\,(\mathsf{ug} \wedge \mathbf{X}_{\{2,3\}}\mathsf{true})$ with probability 8.4092e-3, this is greater than the one for Φ when observation 5 was dropped and 0, 1, 2, 3 and 7 kept.

The formula Φ trivially holds in state 4, since this state satisfies $\mathsf{ug} \wedge \mathbf{X}_{\{2,3\}}\mathsf{true}$ with probability 0.9855. If we take the formula $\mathsf{ug} \wedge \mathbf{X}_{\{0,1,4,5,6,7,8,9,10,11,12\}}\mathsf{true}$, the probability of satisfaction of state 4 is 1.4450e-2. This indicates that state 4 is exceedingly likely to generate observations 2 or 3.

Therefore, the observation 5 is predominantly produced in state 3. Once state 4 is reached, the observations we mostly see are 2 or 3.

5.3.2 Safety properties

A safety property establishes that a *bad thing* does no occur, or does occur with low probability, during the execution of the system or program. Next we offer various safety properties related to the handover scenario run on BERT2.

- *With probability less than 0.05,* BERT2 *releases the object without the user grabbing it.* Again, we use the probabilistic operator. Certainly, BERT2 must be holding the object, and when he does not have it anymore, the user is not grabbing it either. Thus we have the POCTL* formula

  $$\mathcal{P}_{<0.05}(\texttt{rh} \wedge \texttt{rh}\,\mathcal{U}(\neg\texttt{rh} \wedge \neg\texttt{ug})).$$

  The model checker is given this formula and the `handover.poctl` file, it returns $[1, 2, 3, 4]$. This indicates that with a good degree of confidence we expect BERT2 not to drop the object once he has grabbed it.

- *With probability less than 0.05,* BERT2 *abandons its serving position with the user not grabbing the object.* The serving position is equivalent to BERT2 holding the object. What we want to say is that it is unlikely that BERT2 holds the object and next moves onto another state that is not User grab, this subsequent state could be either Robot not hold or Robot pick up. The POCTL* formula describing this property is

  $$\mathcal{P}_{<0.05}(\texttt{rh} \wedge \mathbf{X}_{\Theta}(\texttt{rnh} \vee \texttt{rpu})).$$

  The states that satisfy this specification, i.e., the output of the model checker, are $[1, 2, 3, 4]$. This shows that having BERT2 holding the object, it is highly improbable that he will stop doing so to return to the initial position or to pick up the object once again.

- From Figure 5.6 we infer that *the sequence of observations, i.e., the sequence of finger positions, $\mathcal{O} = \{4, 5, 12, 0\}$ hardly takes place, say $\mathcal{O}$ is generated with probability less than 0.00001.* Formally, we have the specification

  $$\mathcal{P}_{<0.00001}(\mathbf{X}_{\{4\}}(\mathbf{X}_{\{5\}}(\mathbf{X}_{\{12\}}(\mathbf{X}_{\{0\}}\,\mathsf{true})))).$$

  The answer given by the model checker is $[1, 2, 3, 4]$, which means that for every state of the model, $\mathcal{O}$ is unlikely to be generated. Moreover, thanks to the function `trace`, we can find out the actual probability of satisfaction of each state in S for the previous specification. It turns out that the outcome is 0 for all states. This is expected since such sequence of observations can not be drawn from of the handover interaction, as seen in Figure 5.6.

This chapter discussed a real-world application of the model checker for POCTL* implemented in Chapter 4. The situation studied was a human-robot interaction (HRI) in which a robot, namely BERT2, hands an object to a person. We started this chapter by describing BERT2's infrastructure. Next, we shown how to obtain an initial estimation of the HMM $\mathcal{H}$ modelling this HRI. Special attention was paid on reducing the number of observations, objective accomplished by means of the vector quantisation technique. Later, the two adjusting methods offered in Chapter 1 were applied on $\mathcal{H}$ to get the final HMM. We finished this chapter by designing several liveness and safety properties related to the handover process. These properties took the form of POCTL* formulas that, together with $\mathcal{H}$, were supplied to our model checker, whose outputs were discussed and analysed accordingly.

Chapter 6

Conclusions and Future Work

In this thesis we successfully implemented a model checker for POCTL*. We used a slightly modified version of the model checking algorithm described in [35, 36]. We had to make changes to the original POCTL* grammar to explicitly add Boolean values and connectives, as well as the unbounded until operator. Noticeably, to reach the final model checking algorithm proposed in Chapter 3, we had to cope with different and sometimes contradictory approaches considered in [6, 17, 35, 36]. We ended up agreeing with a three-stage model checking algorithm that:

- For stage two, the algorithm uses the measure Pr_s over paths of the HMM $\mathcal{H}$ starting at state s, such that Pr_s incorporates the initial distribution π to its calculation. Similarly, the counterpart measure over paths of the DTMC $\mathcal{D}$, obtained by transforming $\mathcal{H}$, also considers the initial distribution to compute its value. This approach is followed by Courcoubetis *et al.* in [6]. Nevertheless, Zhang [35, 36] and Kwiatkowska [17] do not take the initial probability to compute the measure Pr_s.

- When executing stage three, the algorithm finds the probability of satisfaction of state (s, o) of the DTMC $\mathcal{D}$ through that of the states $\big((s, o), \xi\big)$ and $\big((s, o), \neg\xi\big)$ of the DTMC $\mathcal{D}'$, which is generated according to the construction $C_{\mathbf{X}}$, $C_{\mathcal{U}}$ or $C_{\mathcal{U}^{\leq n}}$. In contrast, Courcoubetis originally calculates the probability of satisfaction of the entire model $\mathcal{D}$ in terms of that of the entire model $\mathcal{D}'$, as explained in [6].

Furthermore, in Chapter 3 we provided additional reasoning and original proofs for some results. These aspects were seemingly omitted or unattended by the papers where the model checker initially appears. Surprisingly, there is no treatment for the bounded until operator $\mathcal{U}^{\leq n}$ in [6]; therefore, we explicitly added the missing construction $C_{\mathcal{U}^{\leq n}}$ to stage three of our algorithm.

The Haskell implementation was devised in Chapter 4. There were times when the coding process was straightforward, e.g., choosing the representation of HMMs and DTMCs, and calculating the Cartesian product required by stage two of the algorithm to obtain the new set of states. However, there were other times when the implementation was cumbersome, e.g., creating new atomic propositions by constantly carrying an integer that was appended to a string that represented a proposition variable. Importantly, challenging circumstances were overcome such as the apparent need to compute the power of the set of observations, and the coding of a loop statement in a functional language.

The second goal of this thesis was also achieved, i.e., we explained how we can operate the model checker for POCTL* to check liveness and safety properties of the handover task from a robot to a user. The robot considered was BERT2 which is part of the *Bristol Robotics Laboratory.* It features high specialised anthropomorphic hands. We focused on his left hand and after numerous handover experiments, we concluded that two joints are specially significant for this task, namely the index and middle finger metacarpophalangeal joint motor current values. These two values formed the pairs that were taken as observations of the HMM modelling this interaction. However, the number of possible observations was not plausible to be handled by the model checker. To make our model checker run in a realisable amount of time, a technique called vector quantisation was used to bring down the number of observations to just thirteen. Consequently, the interpretation given to the observations is that they are specific finger positions. Next, the HMM parameters were adjusted by the Baum-Welch method, this reestimation was confirmed by the K-means algorithm. With all this, we were able to come up with a trained HMM that can be taken as input of our model checker.

Finally, thanks to the model checker for POCTL*, we verified relevant liveness and safety properties, by them we conclude that for this handover process:

- ▷ To release the object held by BERT2, the user has to grab it first. Otherwise BERT2 is remarkably unlikely to let the object go, that is, BERT2 is unlikely to drop the object.
- ▷ The model is likely to generate some observation sequences, e.g., $\mathcal{O} = \{o_0, o_1, o_2, o_3\}$ where the first two observations are taken from $\{2, 3, 5\}$ and the last two are any of the set $\{2, 3, 10\}$; whereas other sequences have no possibility at all to be seen, e.g., $\mathcal{O} = \{4, 5, 12, 0\}$.
- ▷ The sets of typical observations generated by states 3 and 4 are {5} and {2, 3}, respectively. Accordingly, with high probability the critical transition, i.e., when the model goes from state 3 (Robot hold) to state 4 (User grab), is expected to be seen as a sequence of observations 5 followed by observations 2 or 3.

Therefore, we can regard the basic handover interaction on BERT2 as trustworthy.

Now we turn our attention to examine several considerations to extend the work of this thesis. First, we seek improvements to the implementation of the model checker for POCTL*. Zhang *et al.* say in [35, 36] that there are alternative ways to efficiently compute the probability of satisfaction for specific type of formulas, such as:

- Let $\phi = a_{s_0} \wedge \mathbf{X}_{o_0}(a_{s_1} \wedge \mathbf{X}_{o_1}(\ldots(a_{s_n} \wedge \mathbf{X}_{o_n}\mathsf{true})\ldots))$ be a path formula, where for each state $s_i \in S$ we create the atomic proposition a_{s_i} that is true only in s_i. Clearly, every path ω that satisfies it is a member of the basic cylinder set $C\big((s_0, o_0), \ldots, (s_n, o_n)\big)$. Thus, to know whether $s \models \mathcal{P}_{\bowtie p}(\phi)$, it suffices to check whether the measure of this basic cylinder set, i.e., $\Pr_{s_0}(C\big((s_0, o_0), \ldots, (s_n, o_n)\big))$, meets the bound $\bowtie p$.

- Let $\phi = \mathbf{X}_{o_0}\mathbf{X}_{o_1} \ldots \mathbf{X}_{o_n}\mathsf{true}$ be a path formula. To check whether $s \models \mathcal{P}_{\bowtie p}(\phi)$, that is, $\Pr_s\{\omega \in \mathsf{Path}_s \mid \omega \models \phi\} \bowtie p$, we realise that the required measure value can be seen as the probability of the observation sequence $\mathcal{O} = o_0, o_1, \ldots, o_n$ and state $s = S_i$ at time 0, given the model $\mathcal{H}$, i.e., $P[q_0 = S_i, \mathcal{O} \mid \mathcal{H}]$. This probability is efficiently computed by the expression $\alpha_0(i) \cdot \beta_0(i)$ that involves the forward and backward variables studied in Section 1.2.1.

- In Appendix C.2 the logic PCTL is defined. In [17] we find its model checking algorithm, which works far more efficient that the one for POCTL* regarding the until operator and its bounded version. Hence we can use it to deal with QOS formulas of the form $(\varphi_1 \mathcal{U} \varphi_2, \bowtie p)$ and $(\varphi_1 \mathcal{U}^{\leq n} \varphi_2, \bowtie p)$, where the POCTL* path formulas φ_1 and φ_2 are verified recursively.

The previous suggestions can be spotted by the parser of our model checker for POCTL*. Next, the necessary operations should be computed accordingly.

Furthermore, remember that stage two of the model checking algorithm, depicted in Section 3.2, describes the way to construct a DTMC $\mathcal{D} = (S^{\mathcal{D}}, A^{\mathcal{D}}, L^{\mathcal{D}}, \pi^{\mathcal{D}})$ from the original HMM $\mathcal{H} = (S, A, \Theta, B, L, \pi)$. Notice how the probability matrix $A^{\mathcal{D}}$ is defined by $A^{\mathcal{D}}((s, o), (s', o')) = a_{s\,s'} \cdot b_{s'}(o')$. Since the observation o is not part of the right-hand side, the values of $A^{\mathcal{D}}((s, o), (s', o'))$ are the same for all o. In other words, the rows of $A^{\mathcal{D}}$ labelled (s, o) are the same for all $o \in \Theta$. Consequently, we could get a smaller matrix $A^{\mathcal{D}}$ of size $|S| \times |S| \cdot |\Theta|$ where the repeated rows are removed. Likewise, in Section 3.3 stage three of the algorithm is explained together with the constructions $C_{\mathbf{X}}$ and $C_{\mathcal{U}}$ that defined a new DTMC $\mathcal{D}'$. If we could devise a way to manage the transition probabilities in $\mathcal{D}'$ such that the occurrence of cells with zero probability is prevented, both the execution time and the memory consumption would decrease.

Certainly, a large amount of observations dramatically affects the performance of our model checker. So we investigated the vector quantisation method that reduces the number of observations in the HMM. As we studied, this method yields a Voronoi diagram whose peculiarities and applications (see [2, 25]) could be related to interesting properties of the system's observations.

With respect to the handover interaction discussed in Chapter 5, we analysed the basic scenario for which we appointed only four broadly states. It is accurately said in [10, 12] that the human engagement and intention plays a marked role in this task. These papers show an extended system where the robot takes into account the expected sequence of actions a user will make when interested in the handover process, that is, *user looks at the cup*, *user looks away* and *user touches the object*. The decision made by the robot also relies on a specific interval of time from the user first looking at the object to the moment he or she finally touches it. If the timing for the handover is within this interval, the process is successfully performed. Thus further work may take this extended system and check liveness and safety properties of it through the POCTL* model checker.

Finally, the model checker for POCTL* might be used to check important properties of other systems modelled by an HMM such as speech recognition, protein modelling, compressed document processing, human gesture recognition and text recognition.

Appendix A

Probability

Here we present technical definitions and ideas related to probability theory. They are taken from [14, 29, 35].

A.1 Properties and elements of probability theory

Definition A.1. Let $\mathcal{X}$ be a set, not necessarily finite. Then a *σ-algebra* Σ is a nonempty collection of subsets of $\mathcal{X}$ such that the following hold:

1. $\mathcal{X} \in \Sigma$.
2. If $A \in \Sigma$, then $(\mathcal{X} \setminus A) \in \Sigma$.
3. If $A_i \in \Sigma$ for all $i \in \mathbb{N}$, then $\cup_{i\in\mathbb{N}} A_i \in \Sigma$.

If S is any collection of subsets of $\mathcal{X}$, we can always find a σ-algebra that contains S, which trivially is the power set of $\mathcal{X}$, $\mathcal{P}(\mathcal{X})$. By taking the intersection of all σ-algebras containing S, we get the smallest such σ-algebra. ▮

Definition A.2. A *measure* P is defined as a nonnegative real function from σ-algebra Σ, i.e., $P : \Sigma \to \mathbb{R}^+$, such that

$$P(\varnothing) = 0,$$

and for any finite or countable sequence of pairwise disjoint sets $A_1, A_2, \ldots$, with $A_i \in \Sigma$ $\forall i \in \mathbb{N}$, such that $(\cup_{i\in\mathbb{N}} A_i) \in \Sigma$, we have

$$P(\cup_{i\in\mathbb{N}} A_i) = \sum_{i\in\mathbb{N}} P(A_i).$$

Moreover, P is said to be a probability measure on Σ if $P(\mathcal{X}) = 1$. ▮

Joint probability and marginalisation

The joint probability is the probability of two events in conjunction. We denote the joint probability of X and Y by $P[X \wedge Y]$ or $P[X, Y]$.

A common way to compute the probability of X, i.e., $P[X]$, is to sum up the joint probabilities over all outcomes Y of a particular experiment or situation, where X and Y are mutually exclusive events. Therefore,

$$P[X] = \sum_{\text{for all } Y} P[X, Y].$$

This process is called *marginalisation*.

Conditional probability, the product rule and independence

When dealing with uncertainty, once we have obtained evidence concerning previously unknown information, we use *conditional or posterior probabilities*. The notation used is $P[X \mid Y]$. Conditional probabilities can be defined in terms of unconditional probabilities. Accordingly,

$$P[X \mid Y] = \frac{P[X, Y]}{P[Y]} \qquad \text{if } P[Y] > 0.$$

From this we can derived the *product rule*

$$P[X, Y] = P[X \mid Y] \cdot P[Y].$$

If one event X has no influence over another Y, then we say that they are independent. Independence between X and Y can be written as any of the equations

$$P[X \mid Y] = P[X], \qquad P[Y \mid X] = P[Y], \qquad P[X, Y] = P[X]P[Y].$$

Moreover, observe that if the original joint probability is already conditioned, as in $P[X, Y \mid \mathcal{W}]$, then by the product rule we have

$$P[X, Y \mid \mathcal{W}] = P[X \mid Y, \mathcal{W}] \cdot P[Y \mid \mathcal{W}].$$

Equivalently,

$$P[X \mid Y, \mathcal{W}] = \frac{P[X, Y \mid \mathcal{W}]}{P[Y \mid \mathcal{W}]}.$$

Bayes' rule

We can state the product rule in two different ways, namely

$$P[X \wedge Y] = P[X \mid Y] \cdot P[Y]$$
$$P[Y \wedge X] = P[Y \mid X] \cdot P[X].$$

Equating the two right hands and dividing by $P[X]$, we get the formula known as *Bayes' rule*

$$P[Y \mid X] = \frac{P[X \mid Y] \cdot P[Y]}{P[X]}.$$

A more general version results when taking the previous probabilities conditioned on some background evidence $\mathcal{W}$, that is,

$$P[Y \mid X, \mathcal{W}] = \frac{P[X \mid Y, \mathcal{W}] \cdot P[Y \mid \mathcal{W}]}{P[X \mid \mathcal{W}]}.$$

Appendix B

Probability Measures over DTMCs

In Chapter 1 we defined a probability measure over HMMs. In this appendix we will do so for DTMCs, following the ideas presented in [6, 17].

A *path* represents an execution of a system described by a DTMC $\mathcal{D} = (S, A, L, \pi)$. A path ω is defined as a non-empty sequence of states $s_0, s_1, \ldots$ where $s_i \in S$ and $a_{s_i\,s_{i+1}} > 0$ for all $i \geq 0$. A path can be either finite (ω^{fin}) or infinite (ω). We denote the $(i+1)$st state of a path ω by $\omega(i)$ and the length of ω, that is, the number of transitions, by $|\omega|$. We say that a finite path ω^{fin} is a prefix of the infinite path ω if $\omega^{fin}(i) = \omega(i)$ for $0 \leq i \leq |\omega^{fin}|$. To indicate the suffix of ω that starts at state s_k, that is, $s_k, s_{k+1}, \ldots$, we write $\omega[k]$. The sets of all finite and infinite paths in $\mathcal{D}$ starting in state s are denoted $\mathsf{Path}_s^{fin,\mathcal{D}}$ and $\mathsf{Path}_s^{\mathcal{D}}$, respectively. When the model $\mathcal{D}$ is obvious from the context, we omit it from the notation.

To study the probabilistic behaviour of a DTMC, we have to determine the probability that certain paths are taken. With that goal in mind, we define, for each state s in S, a measure $\Pr_s$ over Path_s. We adhere to the basic cylinder construction as follows. The *basic cylinder* set $C(\omega^{fin})$ is

$$C(\omega^{fin}) \overset{def}{=} \{\omega \in \mathsf{Path}_s \,|\, \omega^{fin} \text{ is a prefix of } \omega\}.$$

Let Σ_s be the smallest σ-algebra on Path_s (see previous Appendix A.1) which contains all the sets $C(\omega^{fin})$, where ω^{fin} may be any path in Path_s^{fin}. We define $\Pr_s$ on Σ_s as the unique measure such that

$$\begin{aligned} \Pr_s\big(C(\omega^{fin})\big) &= \Pr_s\big(C(s, s_1, \ldots, s_n)\big) \\ &= \begin{cases} \pi_s & \text{if } n = 0 \\ \Pr_s\big(C(s, s_1, \ldots, s_{n-1})\big) a_{s_{n-1}\,s_n} & \text{otherwise,} \end{cases} \end{aligned}$$

where $n = |\omega^{fin}|$. Moreover, we can rewrite the previous probability measure as

$$\Pr_s\big(C(\omega^{fin})\big) = \pi_s \prod_{i=1}^{n} a_{s_{i-1}s_i}.$$

Let Σ be the smallest σ-algebra on Path, i.e., the set of paths of $\mathcal{D}$ starting at any state $s \in S$. We use the measure $\Pr_s$ to define the probability measure $\Pr_{\mathcal{D}}$ over Σ as follows:

$$\begin{aligned}
&\Pr_{\mathcal{D}}\big(\bigcup_{s\in S} C\big(s,\ldots,s_k)\big) \\
=&\sum_{s\in S} \Pr_{\mathcal{D}}\big(C(s,\ldots,s_k)\big) \quad \text{(By Definition A.2)} \\
=&\sum_{s\in S} \Pr_{s}\big(C(s,\ldots,s_k)\big),
\end{aligned}$$

where the family of sets $\bigcup_{s\in S} C(s,\ldots,s_k) \in \Sigma$ consists of disjoint cylinder sets.

Now we are able to quantify the probability that a DTMC behaves in a certain way by applying the associated measure $\Pr_{\mathcal{D}}$ (or $\Pr_s$) over the set of paths which satisfy a particular specification.

Appendix C

Logics: PCTL, LTL and QLS

This appendix is dedicated to show logics that have a remote relation with POCTL* sublogics.

C.1 PCTL

The definition given here follows the lines presented in [30]. PCTL stands for *Probabilistic Computational Tree Logic*, and its syntax is

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

where a is an atomic proposition, $\bowtie \{\leq, <, >, \geq\}$, $p \in [0,1]$ and $k \in \mathbb{N}$.

We use PCTL as a specification language of DTMCs (see Definition 1.1). Additionally, paths and probability measures over DTMCs are defined as explained in Appendix B. We can distinguish between state formulas (Φ) and path formulas (ϕ), which are evaluated over states and paths, respectively.

Let $\mathcal{D} = (S, A, L, \pi)$ be a labelled DTMC. The semantics of PCTL over $\mathcal{D}$ is given through the satisfaction relation $\models$, which is inductively defined next. For any state $s \in S$, we have

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

where for any path $\omega \in \mathsf{Path}_s$

$$\begin{array}{ll} \omega \models \mathbf{X}\Phi & \text{iff } \omega(1) \models \Phi\\ \omega \models \Phi_1\mathcal{U}^{\leq k}\Phi_2 & \text{iff } \exists i \leq k.\ (\omega(i) \models \Phi_2 \wedge \forall j < i.\ \omega(j) \models \Phi_1)\\ \omega \models \Phi_1\mathcal{U}\Phi_2 & \text{iff } \exists k \geq 0.\ (\omega \models \Phi_1\mathcal{U}^{\leq k}\Phi_2). \end{array}$$

C.2 LTL and QLS

The next definition is taken in part from [33] and [35]. Here we explicitly have additional features, such as the false Boolean value, the disjunction operator and the bounded until operator. Moreover, formulas of *linear-time propositional temporal logic* (LTL) are built from a set Prop of atomic propositions and are formed according to the grammar production

$$\phi := \mathsf{true} \mid \mathsf{false} \mid a \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \mathbf{X}\phi \mid \phi\,\mathcal{U}^{\leq k}\phi \mid \phi\,\mathcal{U}\phi,$$

where $a \in \mathsf{Prop}$ and $k \in \mathbb{N}$.

LTL is interpreted over computations. A computation is a function $\rho : \mathbb{N} \to 2^{\mathsf{Prop}}$, which at each instant of time (natural number) assigns truth values to the elements of Prop. For a function ρ and an instant $i \in \mathbb{N}$, we define a satisfaction relation ($\models$) inductively over the LTL formulas as follows:

$$\begin{array}{ll}
\rho, i \models \mathsf{true} & \text{for every function } \rho \text{ and every } i \in \mathbb{N} \\
\rho, i \not\models \mathsf{false} & \text{for every function } \rho \text{ and every } i \in \mathbb{N} \\
\rho, i \models a & \text{iff} \quad a \in \rho(i) \\
\rho, i \models \neg\phi & \text{iff} \quad \rho, i \not\models \phi \\
\rho, i \models \phi_1 \vee \phi_2 & \text{iff} \quad \rho, i \models \phi_1 \vee \rho, i \models \phi_2 \\
\rho, i \models \phi_1 \wedge \phi_2 & \text{iff} \quad \rho, i \models \phi_1 \wedge \rho, i \models \phi_2 \\
\rho, i \models \mathbf{X}\phi & \text{iff} \quad \rho, i+1 \models \phi \\
\rho, i \models \phi_1\mathcal{U}^{\leq k}\phi_2 & \text{iff} \quad \exists j, i \leq j \leq i+k.\ (\rho, j \models \phi_2 \wedge \forall l, i \leq l < j.\ \rho, l \models \phi_1) \\
\rho, i \models \phi_1\mathcal{U}\phi_2 & \text{iff} \quad \exists j \geq i.\ (\rho, j \models \phi_2 \wedge \forall l, i \leq l < j.\ \rho, l \models \phi_1).
\end{array}$$

Moreover, we say that ρ satisfies a formula ϕ, which we write as $\rho \models \phi$, if and only if $\rho, 0 \models \phi$.

We can see computations as infinite words over the alphabet 2^{Prop}, this word turns out to be the concatenation of $\rho(i)$ for every $i \in \mathbb{N}$, i.e., $\rho(0)\rho(1)\rho(2)\ldots$. In [33], it is stated that the computations satisfying a given LTL formula are exactly those accepted by some finite automaton on infinite words.

We defined the logic QLS as a quantitative LTL specification. QLS formulas are pairs $(\phi, \bowtie p)$, where ϕ is a LTL formula, $\bowtie \in \{\leq, <, >, \geq\}$ and $p \in [0, 1]$. In [35] the formula ϕ is interpreted over DTMCs. So let $\mathcal{D} = (S, A, L, \pi)$ be a DTMC and s a state in S, the satisfaction relation $\models$ is given by the rule

$$\mathcal{D}, s \models (\phi, \bowtie p) \iff \Pr_s\{\omega \in \mathsf{Path}_s \mid \omega \models \phi\} \bowtie p.$$

Appendix D

Sample Execution of the Model Checker

This appendix is dedicated to explain step-by-step how the model checker can be executed, as well as to show the format that its input arguments must attain to, they are a `.poctl` file with the HMM of interest and a POCTL* formula.

Once the Haskell compiler *ghci* is running, we have to load the module `Main.hs` that contains the function `main`, which is called to initiate the model checker. Next, the legend: `Enter the file name where the HMM is located.` is shown and the program waits for a `.poctl` file to be passed to it.

D.1 The `.poctl` file

The six elements of an HMM have to be defined within this file. We recall that $\mathcal{H} = (S, A, \Theta, B, L, \pi)$, thus we use the reserved words `States`, `Transitions`, `Observations`, `ObsProb`, `Labelling` and `Initial` to define its respective elements. Moreover, in the `.poctl` file the sets of states and observations are given as integer numbers representing their length, the probability matrices for the state transitions and the observation distributions are expressed as lists of lists of double values, the labelling function is a list of lists of strings and, finally, the initial distribution is a list of double values. For instance, in [35] an HMM is offered in Chapter 5 as part of Example 5.1.1 on how the model checking algorithm for POCTL* works. We code that same HMM in the file `ModelZhang.poctl`, whose content is:

```
States = 5

Transitions = [[0, 0.5, 0, 0, 0.5],
               [0, 0, 0.5, 0, 0.5],
               [0, 0, 0, 0.5, 0.5],
               [0, 0, 0, 0.5, 0.5],
               [0, 0, 0, 0, 1.0]]

Labelling = [["a"],["a"],[""],["b"],[""]]
```

```
Observations = "3"

ObsProb = [[0.33333333333333,0.33333333333333,0.33333333333333],
           [0.33333333333333,0.33333333333333,0.33333333333333],
           [0.33333333333333,0.33333333333333,0.33333333333333],
           [0.33333333333333,0.33333333333333,0.33333333333333],
           [0.33333333333333,0.33333333333333,0.33333333333333]]

Initial = [1, 0, 0, 0, 0]
```

We require the definition of the number of states (identified by the reserved word `States`) to be the first element to be given by this file. The reason is that knowing this quantity helps us to define the other items composing the HMM. Likewise, the number of observations (`Observations`) has to be established before the observation probability matrix (`ObsProb`). Besides these two constraints, the rest of the HMM elements can be defined regardless of their position in the `.poctl` file.

So, when we are asked for a file where the HMM is located, we might answer `ModelZhang.poctl`. As a result, out of this file the program generates a record value of type `HMM` as defined in Chapter 4. Now the input POCTL* formula is requested by the model checker with the sentence: `Enter the POCTL* formula we are interested in.`, projected on the screen.

D.2 Typing POCTL* formulas

In Remark 3.1 of Chapter 3, we fixed the grammar for the POCTL* formulas, and its corresponding `Haskell` implementation is given back in Chapter 4. However, that encoding is not user friendly at all. In order to offer a less cumbersome manner to type these formulas, we associate to each grammar's terminal symbol a more natural encoding according to the next table.

Terminal Symbol	**New encoding**
Boolean value true	Reserved letter `T`
Boolean value false	Reserved letter `F`
Atomic proposition a	Sequence of non-reserved letters
Negation operator $\neg$	Symbol `~`
Disjunction operator $\vee$	Reserved letter `v`
Conjunction operator $\wedge$	Symbol `^`
Probabilistic operator $\mathcal{P}_{\bowtie p}$	`P[`$\bowtie p$`]`, where `P` is a reserved letter
Next operator $\mathbf{X}_{\{m_1,\ldots,m_i\}}$	`X_`$\{m_1,\ldots,m_i\}$, where `X` is a reserved letter
Bounded until operator $\mathcal{U}^{\leq n}$	`U` n, where `U` is a reserved letter
Unbounded until operator $\mathcal{U}$	Reserved letter `U`
Left Parenthesis (	Symbol `(`
Right Parenthesis)	Symbol `)`

Observe that,

- The encoding of the comparison symbol $\bowtie$ can take any of the values `<`, `<=`, `>=` or `>`.
- For the threshold `double` value p used by the probabilistic operator, the inequality $0 \leq p \leq 1$ holds.
- Since we are considering the observations as integer numbers from 1 to the value of `Observations` defined in the `.poctl` file, every element in the subset of observations $\{m_1, \ldots, m_i\}$, attached to the next operator, also lies within that interval.
- The bound n used in $\mathcal{U}^{\leq n}$ is a non-negative integer.

Continuing with Example 5.1.1 found in Chapter 5 of [35], we are interested to knowing whether state s_1 satisfies the formula $(\neg b) \wedge \mathcal{P}_{<0.05}(a \mathcal{U} (\mathbf{X}_{\{1\}} b))$ based on the HMM packed in the file `ModelZhang.poctl`. After being asked for a formula to verify, we type in the terminal `(~b) ^ P[<0.05] (a U (X_{1}b))`. The software runs the model checking algorithm on the already passed HMM and this POCTL* formula, it takes a couple of seconds and returns

```
The states that satisfy it are:
[1,2,3,5]
```

Since s_1 has initial distribution value of 1, we do not regard states 2, 3 and 5. We conclude that $s_1 \models (\neg b) \wedge \mathcal{P}_{<0.05}(a \mathcal{U} (\mathbf{X}_{\{1\}} b))$. This is also the answer shared by [35][11].

Afterwards, the model checker asks whether the user wants to keep verifying properties. So, it prints: `Do you want to continue checking more specifications? y/n:`. If the answer is `y`, the program waits for a new formula to be entered. If we respond with an `n`, the model checker finishes.

Remark D.1. A subtle issue arises when trying to find the states that satisfy the formula Φ following Zhang's method. He does not take into account the initial distribution π to calculate the measure $\Pr_s$, contrary to what we do. We can achieve the same result that Zhang obtains by executing the model checker $|S|$ times, such that for $i = 1, \ldots, |S|$ we have $\pi_{s_i} = 1$ and $\pi_{s_j} = 0$, $\forall i \neq j$. With this consideration, the model checker is executed. If i is in the output, then we mark the ith state as satisfying Φ. At the end of this process we will have identified all states satisfying Φ, just as Zhang does.

By the previous remark, we get states 1 and 5 as the outcome of Example 5.1.1. This is, too, the solution given by [35].

[11] In [35], Zhang uses state indices ranging from 0 to $n-1$. Nonetheless, our implementation specifies state indices in the range 1 to n. Same situation happens with the observation indices. Consequently, Zhang's answer to Example 5.1.1, s_0, is equivalent to ours, i.e., s_1.

Bibliography

[1] Alpern, B. and Schneider, F. B. *Defining liveness.* Information Processing Letters, Vol. 21, No. 4, pp. 181-185. October 1985.

[2] Aurenhammer, F. and Klein, R. *Voronoi diagrams.* In Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, Eds., pp. 201-290, Elsevier Science Publishers B.V. North-Holland, Amsterdam, 1999.

[3] Baier, C. *On Algorithmic Verification Methods for Probabilistic Systems.* Habilitation thesis, Universität Mannheim. 1999.

[4] Barrett, C., Sebastiani, R., Seshia, S. A. and Tinelli, C. *Satisfiability modulo theories.* In Handbook of Satisfiability, A. Biere, M. Heule, H. van Maaren, and T. Walsch, Eds., Chapter 12, pp. 737-797, IOS Press. Amsterdam, 2008.

[5] Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R. and Tacchella, A. *NuSMV 2: An OpenSource Tool for Symbolic Model Checking.* Proceedings of the 14th International Conference on Computer Aided Verification, pp. 359-364, Springer-Verlag. 2002.

[6] Courcoubetis, C. and Yannakakis, M. *The Complexity of Probabilistic Verification.* Journal of the ACM, Vol. 42, No. 4, pp. 857-907. July 1995.

[7] Eisner, J. *An interactive spreadsheet for teaching the forward-backward algorithm.* In Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics, Vol. 1, pp. 10-18. Philadelphia, 2002.

[8] El-Yacoubi, A., Gilloux, M., Sabourin, R. and Suen, C.Y. *An HMM-Based Approach for Off-Line Unconstrained Handwritten Word Modeling and Recognition.* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 21, No. 8, pp. 752-760. August 1999.

[9] Ghani, K. *Adaptive Control Systems for Verifiably Correct Human-Robot Interaction.* Master's thesis, University of Bristol, England. January 2012.

[10] Grigore, E. C. *I Robot, I Think.* Master's thesis, University of Bristol, England. May 2012.

[11] Grigore, E. C., Eder, K., Lenz, A., Skacheck, S., Pipe, A. G. and Melhuish, C. *Towards Safe Human-Robot Interaction.* In 12th Conference Towards Autonomous Robotic Systems (TAROS), pp. 323-335. 2011.

[12] Grigore, E. C., Eder, K., Pipe, A., Melhuish, C. and Leonards, U. *Joint Action Understanding improves Robot-to-Human Object Handover.* In Intelligent robots and systems. IEEE, pp. 1-8. 2013.

[13] Huth, M and Ryan, M. *Logic in Computer Science. Modelling and Reasoning about Systems.* Second edition. Cambridge University Press. 2004.

[14] Jech, T. J. *Set theory, The 3rd millennium edition, revised and expanded.* Springer Monographs in Mathematics, Springer-Verlag. Berlin, 2003.

[15] Krogh, A., Brown, M., Mian, I. S., Sjlander, K. and Haussler, D. *Hidden Markov Models in Computer Biology.* Journal of Molecular Biology, Vol. 235, pp. 1501-1531, Elsevier. 1994.

[16] Kwiatkowska, M. *Model Checking for Probability and Time: From Theory to Practice.* In Proc. 18th IEEE Symposium on Logic in Computer Science (LICS'03), pp. 351-360, IEEE CS Press. Invited paper. June 2003.

[17] Kwiatkowska, M., Norman, G. and Parker, D. *PRISM 4.0: Verification of Probabilistic Real-time Systems.* Proc. 23rd International Conference on Computer Aided Verification (CAV'11), pp. 585-591, Springer. 2011.

[18] Lee, D.-S. *Substitution Deciphering Based on HMMs with Applications to Compressed Document Processing.* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, No. 12. December 2002.

[19] Lee, K. *Context-Dependent Phonetic Hidden Markov Models for Speaker-Independent Continuous Speech Recognition.* IEEE Transactions on Acoustics, Speech and Signal Processing, pp. 599-609. 1990.

[20] Lenz, A., Skachek, S., Hamann, K., Steinwender, J., Pipe, A. G. and Melhuish, C. *The BERT2 infrastructure: An integrated system for the study of human-robot interaction.* Proceedings of the 10th IEEE-RAS International Conference on Humanoid Robots (Humanoids), pp. 346-351. 2010.

[21] Linde, Y., Buzo, A. and Gray, R. M. *An Algorithm for Vector Quantizer Design.* IEEE Transactions on Communications, Vol, 28, pp. 84-95. 1980.

[22] Llarena, A. and Rosenblueth, D. *Model Checking Applied to Humanoid Robotic Soccer.* In Advances in Autonomous Robotics, Vol. 7429 of LNCS, Springer. 2012.

[23] Makhoul, J., Roucos, S. and Gish, H. *Vector Quantization in Speech Coding.* IEEE Proceedings, Vol. 73, pp. 1551-1588. November 1985.

[24] Moura, L. de, and Bjørner, N. *Z3: An efficient SMT solver.* 14th International Conference, TACAS 2008. Held as Part of the Joint European Conferences on Theory and Practice of Software, 2008. Budapest, Hungary. March 29-April 6, 2008. Proceedings, Vol. 4963 of Lecture Notes in Computer Science, pp. 337-340, Springer. 2008.

[25] Okabe, A., Boots, B., Sugihara, K. and Nok Chiu, S. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* Wiley series in probability and statistics. Second edition. John Wiley & Sons, LTD. 2000.

[26] Rabhi, F. and Lapalme, G. *Algorithms; a Functional Programming Approach.* Second edition. Addison-Wesley Longman Publishing Co., Inc. 2006.

[27] Rabiner, L.R. *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition.* Proceedings of the IEEE, Vol. 77, No. 2, pp. 257-286. February 1989.

[28] Rybski, P. E. and Voyles, R. M. *Interactive Task Training of a Mobile Robot through Human Gesture Recognition.* Proceedings of the 1999 IEEE International Conference on Robotics & Automation. Detroit, Michigan. May 1999.

[29] Russell, S.T. and Norvig P. *Artificial Intelligence. A modern approach.* Second edition. Prentice-Hall, Inc. 2003.

[30] Rutten, J. J. M. M., Kwiatkowska, M., Norman, G. and Parker, D. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems (CRM Monograph Series).* American Mathematical Society. 2004.

[31] Stamp, Mark. *A revealing introduction to Hidden Markov Models.* January 2004. `http://www.cs.sjsu.edu/faculty/stamp/RUA/HMM.pdf`

[32] Strunk, E.A., Aeillo, M.A. and Knight, J.C. *A survey of tools for model checking and model based development, Technical Report*, Computer Science Department, University of Virginia, Charlottsville. June 2006.

[33] Vardi, M. Y. *An Automata-Theoretic Approach to Linear Temporal Logic.* In Logics for Concurrency: Structure versus Automata, Lectures Notes in Computer Science, Vol. 1043, pp. 238-266, Springer-Verlag. Berlin, 1996.

[34] Xiaolin L., Parizeau, M. and Plamondon, R. *Training Hidden Markov Models with Multiple Observations-A Combinatorial Method.* IEEE Trans. Pattern Anal. Mach. Intell., Vol. 22, No. 4, pp. 371-377. 2000.

[35] Zhang, L. *Logic and Model Checking for Hidden Markov Models.* Master's thesis, Saarland University, Germany. March 2004.

[36] Zhang, L., Hermanns, H. and Jansen, D.N. *Logic and model checking for hidden Markov models.* In Wang, F. (ed.) Formal techniques for networked and distributed systems, FORTE 2005. LNCS, Vol. 3731, pp. 98-112, Springer. Berlin, 2005.