VERIFICATION OF LINEAR-TIME PROPERTIES FOR FINITE PROBABILISTIC SYSTEMS

BY

DILEEP RAGHUNATH KINI

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Doctoral Committee:

Professor Mahesh Viswanathan, Chair
Professor Gul Agha
Professor Madhusudan Parthasarathy
Dr. Sumit Gulwani, Microsoft Reasearch

Abstract

With computers becoming ubiquitous there is an ever growing necessity to ensure that they are programmed to behave correctly. Formal verification is a discipline within computer science that tackles issues related to design and analysis of programs with the aim of producing well behaved systems. One of the core problems in this domain is what is called the model checking problem: given a mathematical model of a computer and a correctness specification, does the model satisfy the specification? In this thesis we explore this question for Markov Decision Processes (MDPs), which are finite state models involving stochastic and non-deterministic behaviour over discrete time steps. The kind of specifications we focus on are those that describe the correctness of individual executions of the model, called linear time properites. We delve into two different semantics for assigning meaning to the model checking problem: execution based semantics and distribution based semantics.

In the execution based semantics we look at specifications described using Linear Temporal Logic (LTL). The model checking problem under this semantics are of two kinds: qualitative and quantitative. In the qualitative version we are interested in finding out if the specification is satsified with non-zero probability, and in the more general quantitative version we want to know whether the probability of satisfaction is greater than a given quantity. The standard way to do model checking for both cases involves translating the LTL formula into an automaton which is then used to analyze the given MDP. One of the contributions of this thesis is a new translation of LTL to automata that are provably smaller than previously known ones. This translation helps us in reducing the asymptotic complexity of qualitative model checking of MDPs against certain fragments of LTL. We implement this translation in a tool called `Büchifier` to show its benefits on real examples. Our second main contri-

bution involves a new automata-based algorithm for quantitative model checking that along with known translations of LTL to automata, which gives us new complexity results for the problem for different fragments of LTL.

In the distribution based semantics we view MDPs as producing a sequence of probability distributions over its state space. At each point of time we are interested in the truth of atomic propositions, each of which tells us whether the probabilty of being in a certain states is above or below a given threshold. Linear time properties over these propositions are then used to describe correctness criteria. The model checking problem here happens to be undecidable in general, and therefore we consider restrictions on the problem. First we consider propositions which are robust: a proposition is said to be robust when the probability of being in its associated set of states is always well separated from its given threshold. For properties described over such propositions we observe that the model checking problem becomes decidable. But checking for robustness itself is an undecidable problem for MDPs. So we focus our attention on a subclass of MDPs called Markov Chains which exhibit stochastic behaviour without non- determinism. For Markov Chains we show that checking for robustness and model checking under robustness become tractable and we provide an analysis of the computational complexity of these problems.

Acknowledgements

I owe an immense debt of gratitude to my advisor Mahesh, whose constant support has made this thesis possible. I thank him for nurturing my intellectual needs and guiding me through the world of research with unwavering optimism and patience. He has always been a source of inspiration through his articulate ways of teaching. His insightful and determined approach to problem solving combined with his calm nature made it an absolute pleasure to work with him all these years.

Secondly, I would like to thank Sumit Gulwani. My journey with Sumit began with Automata Tutor where he was instrumental in envisioning the project and bringing together multiple researchers to bring to frution one of the most impactful endeavours I have worked on. Sumit also welcomed me to Microsoft Research for pursuing multiple internships during which I learnt a lot from him. His passionate and driven approach to research is something I will always look up to.

I would like to thank all my other collaborators during my Ph.D. which include Rohit Chadha, Loris D'Antoni, Rajeev Alur, Bjorn Hartmann, and Umang Mathur for the numerous discussions and support through different stages of my journey. I would also like to thank Srivatsan, with whom I worked prior to my Ph.D., who encouraged me to pursue research. I am also very thankful to all my friends in Champaign-Urbana who have made me feel like family for the past few years. Finally, I would like to thank my parents who have been supportive of my decision to pursue higher education in a place far away from home.

Table of Contents

Chapter 1 Introduction 1

Chapter 2 Execution Based Semantics 14

Chapter 3 Monitors for Safety LTL specifications 21

Chapter 4 Qualitative Model Checking 41

Chapter 5 Quantitative Model Checking 68

Chapter 6 Experiments with Büchifier 81

Chapter 7 Distribution Based Semantics 85

Chapter 8 Complexity of Checking Robustness 95

Chapter 9 Decidability under Limit Robustness 106

Chapter 10 Conclusion 110

References 113

Appendix A Details of LDBA Construction 120

Chapter 1

Introduction

1.1 Motivation

Software systems have become ubiquitous and control various aspects of human life. Computers have widespread applications, from health-care to transportation, from banking to telecommunications, which critically depend on the programs behaving correctly. Formal methods has emerged as the science of designing and analyzing programs with an aim of producing well behaved computer systems. A corner stone of this area is what is called the *model checking* paradigm, pioneered by Clarke, Emerson and Sistla [15, 14] and Queille and Sifakis [47]. Clarke, Emerson and Sifakis received the Turing Award in 2007 for their seminal work. Model checking involves considering a mathematical model of the system and its requirements, and then automatically checking whether the system satisfies those requirements. Thus a model checking algorithm takes two inputs:

- A model: a design of the system that describes the *how* the given system behaves.
- A requirement: a property that specifies *what* the system is supposed to do.

The models of primary interest in this thesis are those that describe the execution or evolution of the system as a sequence of configurations over discrete time steps. A linear-time property specifies what it means for a sequential execution to be correct. Some examples of linear-time properties described in plain English are: *"Every message that is received is eventually acknowledged"* or *"It is always the case that at most one process has acquired a certain lock"*. In these examples note the usage of *eventually* and *always* which refer to passage of time. These properties are temporal in nature as they relate to the configuration

of the system at different time-points within a single execution. Thus the model checking problem becomes one of ensuring that executions of the system, of which there might be infinitely many, meet the specification.

In this thesis we consider systems that evolve probabilistically. Probabilities are used in many crucial ways in computational systems:

- Randomized algorithms exploit randomness to either improve average-case efficiency, or improve efficiency at the expense of accuracy or correctness. Randomness is also used in distributed protocols such as leader election and consensus.
- In physiochemistry applications related to drug administration one is interested in capturing relative concentrations of chemicals like enzymes, drugs and hormones as probabilities and then reason about their evolution and pathways in the animal body.
- Probabilities are used in machine learning to summarize relative likelihood of different patterns in historical data, which is then used to make quantified judgments about future events.
- In performance evaluation, probabilities are used to model information regarding reliability such as a processor's failure rate, a network channel's distribution of time delay. These are then used in computing expectation of various quantities of interest like queue length, waiting time, time to recovery, load per processor etc.

When considering models that have probabilities it is natural that some executions of the system might be more or less likely than others. So the correctness might not be of universal/existential in nature as for non-probabilistic systems. For example there might be an execution of the system that is bad according to the specification, but the probability of that particular execution occurring might be zero or small enough that it does not matter. Here the correctness becomes *quantitative* in nature: one could possibly ask *"What is the probability that a random execution of the system satisfies the specification?"*, *"Is that probability greater than some prescribed threshold?"*. Or correctness could be about comparing probabilities of the system being in certain state(s) at different time points. For example:

"Is the drug concentration in the liver always greater than 0.5 grams during the day?". Here the drug concentration (modeled as probabilities) in different parts evolves over time in the system. The correctness questions that one can pose depends on the semantics of the probabilistic system one chooses. But first we look at the kind of probabilistic systems we focus in this thesis.

In this thesis we investigate questions related to the probabilistic model called *Markov Decision Processes* (MDPs). MDPs are finite-state transition systems which have a probabilistic and a non-deterministic component. This makes them a convenient formalism for describing concurrent stochastic programs with finite states. A *scheduler* is used to define semantics for an MDP. A scheduler resolves the non-deterministic aspect of the MDPs behaviour (we will provide precise definitions in the subsequent section). The model checking problem then becomes: *"Is there a scheduler under which the requirement is violated?"* or *"Is it the case that every scheduler satisfies the requirement?"*. To answer these questions we need to know what does it mean for a specification to be violated/satisfied under a given scheduler. In this thesis we consider two different ways of assigning this meaning: (i) Execution based semantics, (ii) Distribution based semantics. Informally, the difference between the two lies in the way in which the temporal and probabilistic aspects interact to give rise to the meaning of term violation/satisfaction. In the former, a scheduler associates probabilities with the sets of executions, and we are interested in knowing how large or small is the probability measure of the good executions, i.e., executions deemed to be good according to the specification. In the latter, a scheduler yields a sequence of probability distributions over the states, and we are interested in whether this sequence is good. We divide the thesis into two parts, in the first one (Chapters 2 to 6) we explore questions related to the execution based semantics and in the second part (Chapters 7 to 9) we look at the distribution based semantics.

1.2 Formal Background

1.2.1 Transitions Systems, Markov Chains and MDPs

In this section we look at the formalisms we use for modeling a system and writing correctness specifications. We first introduce transition systems for modeling programs that do not have probabilities and then see how we can enrich them to obtain MDPs. Transitions systems are directed graphs where the vertices represent *states* of the system and edges represent *transitions* or change of state. Informally, a state is an instantaneous snapshot of the system. The state gives us all the information needed to know the subsequent evolution of the system. Transitions represent how the state is allowed to change from one instant to the next.

Formally, a transition system is a tuple $(Q, Act, Trans, i)$ where Q is the set of states, Act is a set of actions (names for individual transitions), $Trans \subseteq S \times Act \times S$ is the transition relation and $i \in Q$ is the initial state. For example Figure 1.1 shows a transition system modeling a traffic light system with states $Q = \{R, Y, G\}$ which correspond to red/yellow/green, actions being $Act = \{\alpha, \beta, \gamma\}$, transitions $Trans = \{(R, \alpha, G), (G, \beta, Y), (Y, \gamma, R), (R, \beta, R)\}$ (as indicated by the directed edges between states) and R is the initial state (indicated by the arrow pointing to it).

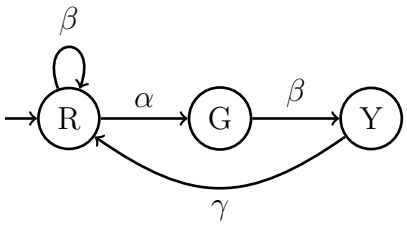


Figure 1.1: Example of a transition system.

A transition system behaves as follows. At any point the system has a *current* state. The next state is decided by looking at the available actions at the current state. So if the current state above is R then the available actions are α and β. An action is chosen *non-deterministically* from the available actions, and it is followed to update the current

state. So, if α is picked from R then the next state is G, and if β is picked from R then the next state is R. The process is repeated from the updated current state. An execution is a sequence of such repeated moves starting from the initial state. Transition systems are a convenient formalism for systems such as traffic lights, communication protocols, digital circuits, concurrent programs etc.

Next, we look at Markov Chains which can be seen as transition systems with probabilistic choice instead of non-determinism. From here onwards, we use $Dist(S)$ to denote the set of all probability distributions over a finite set S. A Markov Chain is a tuple (Q, δ, μ_0) where Q is a finite set of states, $\delta : Q \to Dist(Q)$ is the probabilistic transition function and $\mu_0 \in Dist(Q)$ is the initial distribution. A Markov Chain works as follows: the first state s_0 of an execution is picked by sampling according to μ_0. The next state is picked by sampling $\delta(s_0)$ to obtain s_1, and this process is continued with s_1 and so on. Figure 1.2 presents an example of Markov chain. As you can see instead of actions the transitions are annotated with numbers in the range $[0, 1]$ which add up to 1 for all transitions leaving any particular state. The probabilistic aspect of Markov chains makes it convenient for modeling systems that have uncertainty or randomized algorithms for example election protocols, ethernet protocols, systems of chemical reactions etc.

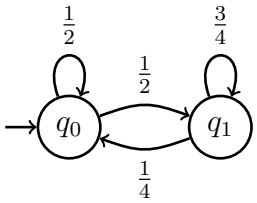


Figure 1.2: An example of a Markov Chain.

Next, we look at Markov Decision Processes (MDP) which combines both non-determinism and probabilistic aspects. A MDP is a tuple (Q, Act, Δ, μ_0) where

- Q is a finite set of states
- Act is a finite set of actions
- $\Delta : Q \times Act \to Dist(Q)$ is a function representing probabilistic transitions

- $\mu_0 \in Dist(Q)$ is the initial distribution

A MDP works as follows: it begins by choosing a state s_0 obtained by sampling the distribution μ_0. After that an action $\alpha_0 \in Act$ is chosen non-deterministically which yields the distribution $\Delta(s_0, \alpha_0)$ which is then sampled to obtain the next state s_1. The process is repeated with s_1. This yields an execution of the MDP of the form $s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \cdots$. As an example we take a look at a randomized mutual exclusion protocol modeled as a MDP in Figure 1.3.

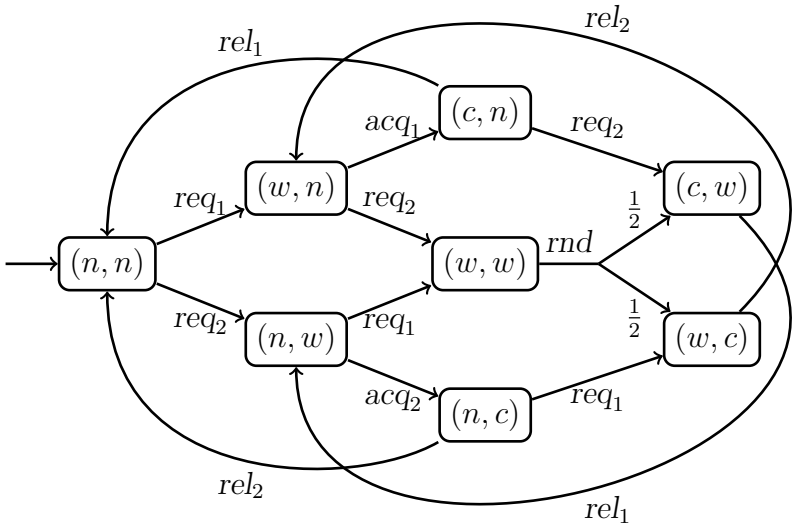


Figure 1.3: MDP for randomized mutual exclusion.

The scenario is one in which two processes are competing for exclusive access to a shared resource. Each process can be in one of 3 states n, w, or c denoting "not using resource", "waiting for resource", "using the resource (critical section)". respectively. The state of the system is represented using a pair (s_1, s_2) denoting the state of the individual processes. A process moves from n to w when it has requested for the resource (action req_i), and moves from w to c when the request has been granted (action acq_i). A process then moves from c

to n when it releases access to the resource (action rel_i). Note that the transitions of these processes can interleave which gives rise to the different states. The system is never in a state (c, c) thus maintaining mutual exclusion. Randomization here is used at only one state (w, w), i.e., if both processes are waiting to gain access then one of them is randomly chosen with equal probability (action *rnd*). An important question one can ask is "What is the probability that if a process requests access at any point then it eventually gains access to the resource?". The answer here is 1 and can be argued as follows. Suppose the first process requests access and is never granted this has to mean that the second process continually requests access and gains access . For this to happen the system has to go through state (w, w) infinitely often and each time the random choice has to end up giving access to the second resource, the probability of this happening is 0. Although we have argued this by hand, it becomes harder to do so when the systems become larger and complicated, which motivates for having efficient algorithms to do it.

1.2.2 The specification language

In the previous example we looked the property "If a process requests access at any point then it eventually gains access". This is an example of a temporal property because it relates states of the system at different time points. A temporal property classifies sequences (over an appropriate domain) as true or false. To elaborate on this we first look at propositions. A proposition is a symbol that can take a value of true/false. The truths of propositions are allowed to change over time. Propositions are usually designed to abstract out a certain characteristic of the system at a given time. For example let the proposition r denote the fact that a request is made at a point and let proposition g denote the fact that a request is granted at a point. We can now abstract the execution of a system into a sequence of truth values (true/false) of these propositions. Consider the finite sequence $\{r\}, \{\}, \{g\}, \{r, g\}$. This sequence is saying that at the first instant in the execution a request was made but access not granted, at the second instant neither a request was made nor was any access granted. Access was granted at the third instant but no request was made (we can infer that this was in response to the request at first instant). At the fourth instant another request was made which was granted access immediately. We will be interested in never-ending executions of

the system and hence infinite sequences of truth assignments to propositions. Let AP denote a finite set of atomic propositions or simply propositions. We use 2^{AP} to denote the set of Boolean assignments to these propositions. The set of all infinite sequence of assignments is then denoted by $(2^{AP})^\omega$. A temporal property P over atomic propositions AP is any subset of $(2^{AP})^\omega$, i.e., $P \subseteq (2^{AP})^\omega$. An infinite sequence $w \in (2^{AP})^\omega$, also called as word, is said to satisfy property P if $w \in P$. The word w is said to violate (not satisfy) property P if $w \notin P$. The usage of such properties in verification will be clarified in the next section. In this section we shall restrict our attention to how such properties are specified formally. Note that individual elements of P are infinitely long strings and P itself can have uncountably many elements. If we are to provide a property as input to the model checking algorithm we need a finite representation of it. This means we can only consider properties that can possibly have finite representations. One such class of well studied properties are what are called ω-regular properties. ω-regularity was first studied in the 1960s by Büchi [10], Muller [44] in connection with logic and network theory, and now the theory of ω-regularity is a field of study in its own right [26]. One way of characterizing ω-regular properties is using *finite-state automata* that process words of infinite length. An automaton is a transition system with an *acceptance condition*. An automaton processes words (sequences over a finite alphabet) from left to right to produce run(s), and the acceptance condition tells us whether a run is good or bad which is used to define the property associated with the automaton. Below is the definition of an automaton with Büchi acceptance condition and the language associated with it. When the alphabet used by an automaton is 2^{AP}, it defines a language which is a temporal property over propositions AP.

Definition 1.1 (Büchi Automata). *A nondeterministic Büchi automaton (NBA) over input alphabet Σ is a tuple (Q, δ, I, F) where Q is a finite set of states; $\delta \subseteq Q \times \Sigma \times Q$ is a set of transitions; $I \subseteq Q$ is a set of initial states and $F \subseteq Q$ is a set of final states.*

A run of a word $w \in \Sigma^\omega$ over a NBA is an infinite sequence of states $q_0 q_1 q_2 \ldots$ such that $q_0 \in I$ and $\forall\, i \geq 0\ (q_i, w_i, q_{i+1}) \in \delta$. A run is accepting if $q_i \in F$ for infinitely many i.

The language accepted by an NBA $\mathcal{A}$, denoted by $L(\mathcal{A})$ is the set of all words $w \in \Sigma^\omega$ which have an accepting run on $\mathcal{A}$.

One way to characterize ω-regular properties is to define it as the class of all languages recognized by non-deterministic Büchi automata. An important subclass of ω-regular properties are those definable using the logic called Linear Temporal Logic (LTL) first introduced by [46], which will be the focus of much of this thesis. LTL builds upon propositional logic by adding *temporal connectives* that relate the truth of propositions at different points. LTL formulae are constructed using atomic propositions from AP, boolean connectives *conjunction* ($\wedge$), *disjunction* ($\vee$), *negation* ($\neg$), temporal connectives *always* ($\mathbf{G}$), *eventually* ($\mathbf{F}$), *until* ($\mathcal{U}$), *release* ($\mathbf{R}$), and *next* ($\mathbf{X}$).

We first describe notational conventions before defining the syntax and semantics of the logic. We use w to denote infinite words over a finite alphabet. We use w_i to denote the i^{th} (index starting at 0) symbol in the sequence w, and use $w(i)$ to denote the suffix $w_i w_{i+1} \dots$ of w starting at i. We use $w[i,j]$ to denote the substring $w_i \dots w_{j-1}$. We use $[n]$ to denote all non-negative integers less than n that is $\{0, 1, \dots, n-1\}$. We begin by recalling the syntax of LTL:

Definition 1.2 (LTL Syntax). *Formulae in* LTL *are given by the following syntax:*

$$\varphi \quad ::= \quad p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \varphi \,\mathcal{U}\, \varphi \mid \varphi \,\mathbf{R}\, \varphi \qquad p \in P$$

Next, we look at the semantics of the various operators. Here the notation $w \vDash \varphi$ indicates that the word w satisfies the property φ.

Definition 1.3 (Semantics). LTL *formulae over a set P are interpreted over words w in $(2^P)^\omega$. The semantics of the logic is given by the following rules*

$$\begin{aligned}
w \vDash p\ (\neg p) &\iff p \in w_0\ (p \notin w_0) & w \vDash \mathbf{X}\varphi &\iff w(1) \vDash \varphi \\
w \vDash \varphi \vee \psi &\iff w \vDash \varphi \text{ or } w \vDash \psi & w \vDash \mathbf{F}\varphi &\iff \exists\, i : w(i) \vDash \varphi \\
w \vDash \varphi \wedge \psi &\iff w \vDash \varphi \text{ and } w \vDash \psi & w \vDash \mathbf{G}\varphi &\iff \forall\, i : w(i) \vDash \varphi \\
w \vDash \varphi \,\mathcal{U}\, \psi &\iff \exists\, i : w(i) \vDash \psi, \text{and} & w \vDash \varphi \,\mathbf{R}\, \psi &\iff \forall\, i : w(i) \vDash \psi, \text{or} \\
&\qquad\quad \forall\, j < i : w(j) \vDash \varphi & &\qquad\quad \exists\, j < i : w(j) \vDash \varphi
\end{aligned}$$

The semantics of φ, denoted by $[\![\varphi]\!]$, is defined as the set $\{w \in (2^P)^\omega \mid w \vDash \varphi\}$.

As an example consider the formula $p \mathcal{U} q$. Any word that satisfies the formula has a point such that q holds at that point and p holds at every point preceding it. For a more complicated formula $\varphi \mathcal{U} \psi$, we have the truth of φ and ψ at each point inductively which we use to define the truth of the bigger formula. A formula $\mathbf{X}\varphi$ holds true for a word if it is true for the suffix starting at position 1. The formula $\mathbf{F}\varphi$ holds true if there is some point along the given word at which φ holds. The formula $\mathbf{G}\varphi$ holds true for a word if φ holds true at every point along the word. The formula $\varphi \mathbf{R} \psi$ holds true if either ψ holds true at every point or φ holds at a certain point, and for all points upto and including that point ψ holds true. Going back to the request-response specification introduced earlier in this section which said "every request for access is eventually granted" can be described using the formula $\mathbf{G}(r \Rightarrow \mathbf{F}g)$, here $\Rightarrow$ is a symbol used for propositional implication.

In this thesis we shall often look at various fragments of LTL. In order to facilitate such discussion we use the notation LTL$(op_1, \ldots, op_k)$ to denote the fragment of LTL that is constructed using boolean combinations of formulae that only use operators $op_1, \ldots, op_k$.

1.3 Contribution

In this section we briefly highlight the main contributions of this thesis, and include other related results in the respective chapters.

In the execution based semantics we consider labeled MDPs where the states are labeled by propositions, and we consider Linear Temporal Logic (LTL) specifications over these propositions. We investigate two versions of the model checking problem:

- Quantitative Model Checking: Given a labeled MDP, an LTL formula describing the bad executions, and $\theta \in [0, 1]$ is there a scheduler under which the probability of formula being satisfied is $> \theta$?

- Qualitative Model Checking: Given a labeled MDP, and an LTL formula describing the bad executions is there a scheduler under which the probability of formula being satisfied is > 0 ?

Observe that in the quantitative version the threshold θ is given as input and can be thought

of as part of the specifications or requirement, and in the qualitative version this threshold is 0 making it a special case of the former. In order to solve these problems we take the automata theoretic approach, which has been used for the non-probabilistic case [59] and also for the probabilistic version [19]. This approach broadly involves, taking the cross product of the model and an automaton derived from the LTL formula, and then doing an analysis of this product. For the quantitative question we show that for certain fragments of LTL the analysis step of the approach can be improved, thus yielding better upper bounds. For the qualitative question we show a new translation for large fragments of LTL to appropriate automata that is provably better than existing methods by an exponential factor, thus giving as much gains in the algorithmic complexity. We provide experimental results for this translation in a prototype implementation, `Büchifier` that constructs smaller automata when compared to state of the art techniques.

In the distribution based semantics we consider labelings on the probability distribution of states. The propositions here correspond to whether the probability of being in a certain set of states is above a threshold. We then consider ω-regular properties over these propositions, defined using automata over infinite strings, as the linear-time specification. The model checking question we investigate here is whether there exists a Markovian scheduler (a scheduler whose decision depends only on the number of steps that have been taken so far) that induces a sequence of distributions which satisfies the ω-regular property. The problem for such labelings turns out to be undecidable in general, so we consider a restriction on the labelings, called robustness, which makes the problem decidable. We analyze the complexity of checking robustness and also of the model checking problem under robustness for MDPs and Markov chains.

1.4 Related Work

The model checking problem for probabilistic programs under the execution based semantics was first investigated by Vardi [58]. Vardi considered the qualitative version of the problem which asks if executions of the probabilistic program satisfy the temporal property with probability 1. They showed that the complexity of this problem for sequential

probabilistic programs (Labeled Markov chains) is PSPACE-complete when specifications are LTL formulae. Courcoubetis and Yannakakis [19] later showed that the complexity of this problem for concurrent probabilistic programs (MDPs) is 2EXPTIME-complete. In this thesis we consider the same problem but restricted to fragments of LTL, and show that complexity can be reduced to EXPTIME and also prove a matching lower bound. In [19], they showed that for Markov Chains the quantitative problem can be solved in PSPACE, using an approach not involving automata. The quantitative version for MDPs was solved by Courcoubetis and Yannakakis [18] using an automata theoretic approach that involves solving a linear program obtained from the cross product of the model and the automaton for the specification.

The distribution based semantics was first considered by Kwon and Agha [40] in which they consider labelings whose truth was derived by looking at linear inequalities over the distribution of states. The labelings we consider are a special case of this, where the only allowed linear combinations correspond to sets of states. But the models considered in [40] are Markov chains that are ergodic and diagonalizable, whereas our decidability results apply to Markov chains in their full generality. The idea of [40] was extended to MDPs by Korthikanti et al. [33], where they show decidability results for two different restrictions: first they show that when the linear labels are such that the constant in the linear inequalities is 0 the problem becomes decidable in PSPACE, and secondly they consider restrictions on regular Markovian schedulers termed "almost acyclic" to prove decidability for general labelings. In our work we consider only one set of schedulers, these are all possible Markovian schedulers. The work of [33] is continued in [13] where they consider special class of MDPs called semi-regular for which they prove decidability of the model checking problem against any ω-regular property.

In this thesis we only consider specifications which are linear time properties, either expressed as LTL formulae or more generally ω-regular languages. LTL was proposed by Pnueli [46] for analyzing non-probabilistic programs which was later adapted for the probabilistic case. Branching time properties form another important class of temporal properties. The concept of time they consider is one which has a branching tree structure as opposed to a sequence of time steps. Computational Tree Logic (CTL) proposed by Clarke and Emer-

son [16] was the first branching time logic to be considered, and it has been followed up with several extensions [16, 20, 21]. CTL allows nesting of of explicit path quantification ($\forall$ paths π starting at state s: π satisfies..., or $\exists$ path π starting at state s: π satisfies...) to express properties that LTL cannot. PCTL is an extension of CTL where the quantification is probabilistic (the probability of paths π starting at state s to satisfy... is $\lesseqgtr \theta$). PCTL and the related model checking problems were first proposed and investigated by Hansson and Jonsson [28].

In terms of implementations of model checking algorithms for probabilistic systems, PRISM [39] is the most prominent. PRISM supports PCTL model checking as well quantitative verification of linear time properties. Our future goals include providing `Büchifier` as a plugin for PRISM to perform qualitative model checking of MDPs against LTL.

Chapter 2

Execution Based Semantics

In this first part of the thesis (starting from here to Chapter 6) we focus on the execution based semantics of MDPs. We consider labeled MDPs where states are labeled by truth assignments to atomic propositions. We define the semantics through schedulers that resolve non-determinism to yield a probability measure on sets of executions, explained Section 2.1. We will be interested in specifications expressed in Linear Temporal Logic (LTL). The model checking problem we investigate is of deciding if the probability associated with the set of paths conforming to the given LTL formula is greater than a threshold. The threshold can be given as an input $\theta \in [0, 1]$ in which case the problem is called quantitative model checking. Or the threshold can be assumed to be 0 in which case it is called qualitative model checking. In Section 2.2 we outline the automata-theoretic approach to solving this problem that involves translating the LTL formula into different kinds of automata and using those automata to perform model checking. In Chapters 3 and 4 we present new translations of fragments of LTL to automata that can be used in the model checking of Markov chains and MDPs. In Chapter 5 we present a new algorithm for quantitative model checking that exploits existing translations of fragments of LTL to automata. In Chapter 6 we discuss our prototype implementation `Büchifier` that implements our constructions of Chapter 4 and compare it with state of the art tools that generate automata for model checking MDPs.

2.1 Basic Definitions

We recall the definition of a *Markov chain* M, a tuple (Q, δ, μ_0) where Q is a set of states, $\delta : Q \to \mathit{Dist}(Q)$ is the probabilistic transition function, and $\mu_0 \in \mathit{Dist}(Q)$ is the initial distribution. When Q is finite, we will often abuse notation and use δ as a stochastic

matrix where the i^{th} row of the matrix corresponds the distribution associated with s_i where $s_0, \ldots, s_n$ is some fixed enumeration of Q. $\delta(s_i, s_j)$ will be denoting the probability of moving from s_i to s_j. A Markov chain M induces a probability distribution over infinite paths of M. In the definition below we will refer to a sequence of states of the Markov chain as a path of the Markov chain.

Definition 2.1. *Given Markov chain $M = (Q, \delta, \mu_0)$, the probability space associated with it is denoted by $\mathcal{P}_M$ is $(\Omega_M, \mathcal{F}_M, \Pr_M)$ where:*

- *The sample space $\Omega_M = Q^\omega$, the set of all infinite paths in M*
- *For finite paths $\pi \in Q^*$ the cylinder set C_π is defined as*

$$C_\pi = \{\pi' \in Q^\omega \mid \pi \text{ is a prefix of } \pi'\}$$

  $\mathcal{F}_M$ is defined as the smallest σ-algebra containing the set $\{C_\pi \mid \pi \in Q^*\}$

- *$\Pr_M$ is the unique probability measure which assigns the cylinder set C_π with the probability value $\mu_0\delta(s_0, s_1)\delta(s_1, s_2)\ldots\delta(s_{k-1}, s_k)$ where $\pi = s_0, s_1, \ldots, s_k$*

A Labeling of a Markov Chain $M = (Q, \delta, \mu_0)$ is a function $L : Q \to 2^{AP}$ mapping each state to a subset of the atomic propositions AP, or equivalently a truth assignment to the propositions. A trace tr is an infinite sequence of assignments to AP. The trace associated with a infinite path $\pi = q_0, q_1, q_2, \ldots$ is the sequence $t_0, t_1, t_2, \ldots$ where $t_i = L(q_i)$. The probability measure $\Pr_M$ can be extended to sets of traces as follows: Let T be a set of traces then $\Pr_M(P)$ is defined as $\Pr_M(\{\pi \in Q^\omega \mid \mathit{trace}(\pi) \in T\})$.

Next, recall that a *Markov Decision Process* $\mathcal{M}$ is a tuple $(Q, \mathit{Act}, \Delta, \mu_0)$ where Q is a finite set of states, Act is a finite set of actions, $\Delta : Q \times \mathit{Act} \to \mathit{Dist}(Q)$ is a function representing probabilistic transitions, and $\mu_0 \in \mathit{Dist}(Q)$ is the initial distribution.

In the execution based semantics of MDPs, one considers a *scheduler* that resolves the non-determinism. At each state $s \in Q$ the scheduler picks an action $a \in \mathit{Act}$, and then the next state is chosen stochastically according to the distribution $\Delta(s, a)$. The scheduler can make its decision based on the states that have been seen so far. The execution begins by picking a state $q \in Q$ based on the initial distribution μ_0.

Definition 2.2. *A* scheduler $\mathfrak{S}$ *for an MDP* $\mathcal{M}$ *is a function* $Q^+ \to Act$ *which maps finite sequence of states to actions. A*

The set of paths in the MDP induced by the scheduler $\mathfrak{S}$ are referred to as $\mathfrak{S}$-paths. The $\mathfrak{S}$-paths of an MDP $\mathcal{M}$ have a one-to-one correspondence with the paths of a Markov Chain $\mathcal{M}_{\mathfrak{S}}$ that is said to be *induced* by the scheduler. $\mathcal{M}_{\mathfrak{S}}$ is obtained by unfolding the MDP graph along the actions prescribed by the scheduler. For simplicity we shall often write $\Delta(s,a)(t)$ as $\Delta(s,a,t)$.

Definition 2.3. *Given an MDP* $\mathcal{M} = (Q, Act, \Delta, \mu_0)$*, a scheduler* $\mathfrak{S}$ *is said to* induce *the Markov chain* $\mathcal{M}_{\mathfrak{S}} = (Q^+, \delta_{\mathfrak{S}}, \mu_0)$ *where for* $\rho = q_0, q_1, \ldots, q_k$*:* $\delta_{\mathfrak{S}}(\rho, \rho q) = \Delta(q_k, \mathfrak{S}(\rho), q)$.

A labeling on the MDP is defined exactly the same way it is defined for Markov chains. The induced Markov chain $\mathcal{M}_{\mathfrak{S}}$ allows one to assign probabilities to temporal-properties described on the labels of the $\mathfrak{S}$-paths on $\mathcal{M}$. If $P \subseteq (2^{AP})^\omega$ is an ω-regular property over AP the set of propositions used in the labeling, then $\Pr^{\mathfrak{S}}_{\mathcal{M}}(P)$ is defined as $\Pr_{\mathcal{M}_{\mathfrak{S}}}(P)$. The problem we are interested is model checking of MDPs.

Definition 2.4. *The* quantitative model checking problem *is to decide if there exists a scheduler* $\mathfrak{S}$ *such that* $\Pr^{\mathfrak{S}}_{\mathcal{M}}(P) > \theta$ *where MDP* $\mathcal{M}$*, property* P *and threshold* $\theta \in [0,1]$ *are given as inputs. When* θ *is fixed to be* 0 *the problem is known as* qualitative *model checking.*

In this part of the thesis we present new complexity results for the qualitative and quantitative model checking problems where the properties are specified in Linear Temporal Logic (LTL) or its sub-fragments.

2.2 Automata Theoretic Approach

The classical algorithm for model checking non-probabilistic systems involves constructing a non-deterministic automaton (Definition 1.1) for the given property (which describes the bad executions), followed by an emptiness check of the product of the model and the automaton (outlined in Figure 2.1). This idea, invented by Vardi et al. [57], paved the way for an automata theoretic approach to model checking temporal properties. Since then

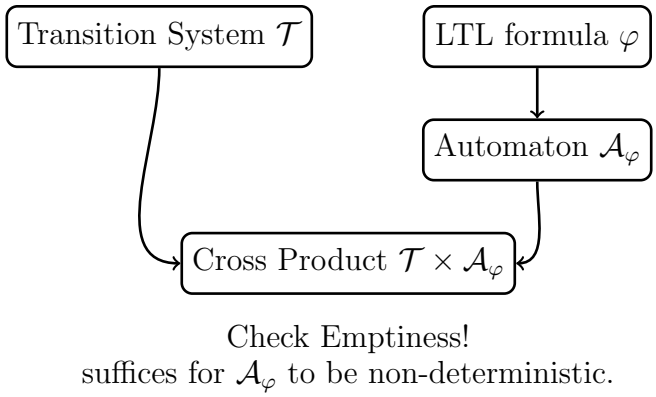


Figure 2.1: Model checking schema for non-probabilistic systems.

there has been a lot of interest in discovering efficient translation of LTL formulae into small automata.

2.2.1 Limit Deterministic Automata

The algorithm for quantitative model checking problem for LTL properties, follows a similar approach. One constructs an automaton for the property P, and solves a linear program obtained from the cross-product of the model (MDP) and the automaton. The solution to the linear program yields the extremal probabilities, $\inf_{\mathfrak{S}} \Pr^{\mathfrak{S}}_{\mathcal{M}}(P)$ and $\sup_{\mathfrak{S}} \Pr^{\mathfrak{S}}_{\mathcal{M}}(P)$, which are then compared to θ to decide the answer. The procedure is outlined in Figure 2.2. The crucial difference here as compared to the non-probabilistic case is that non-deterministic automata don't suffice. One requires the automaton to be deterministic. Deterministic automata are also required for solving games. Hence there has been a lot of effort in the construction of small deterministic automata for LTL. Direct translations of LTL (and fragments of LTL) to deterministic Rabin automata have been proposed [31, 45, 42, 35, 6, 22]. However, any such translation, in the worst case, results in automata that are doubly exponential in the size of the specification [3]; this lower bound holds for any fragment that contains $\wedge$, $\vee$ and $\mathbf{F}$. Courcoubetis and Yannakakis [19] in fact showed that the qualitative model checking problem is itself 2EXPTIME-hard, thus one cannot hope to do any better for the quantitative case than using the LTL to deterministic automata translation. But [19] also showed that for solving the qualitative model checking problem one does not need

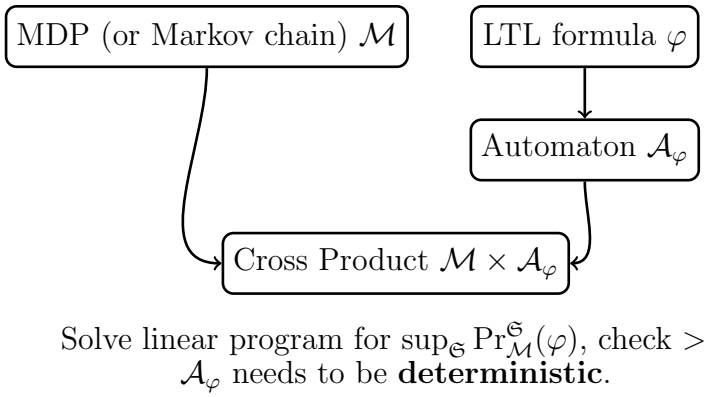


Figure 2.2: Model checking schema for probabilistic systems.

deterministic automata but *limit deterministic* (Definition 2.5) automata suffice (Proposition 2.1). This does not yield an improved upper bound of the qualitative model checking problem for LTL, but is useful because the construction of limit deterministic automata is shown to be simpler than the construction of deterministic automata. In this thesis we show how one can construct limit deterministic automata which are provably more efficient for sub-fragments of LTL. In Chapter 4 we show exponential sized construction for a large class of formulae which we call LTL_{D}. This results in lowering the complexity of the qualitative model checking problem for that fragment.

Definition 2.5 (Limit Determinism). *A NBA (Q, δ, I, F) over input alphabet Σ is said to be* limit deterministic *if for every state q reachable from a final state, it is the case that $|\delta(q, \sigma)| \leq 1$ for every $\sigma \in \Sigma$.*

Proposition 2.1 ([19, Proposition 4.2.3]). *The qualitative model checking problem of MDPs against property specified as a limit deterministic Büchi automata can be solved in time linear in the product of the sizes of the MDP and the automaton.*

2.2.2 Quantitative Model Checking

While qualitative model checking benefits from efficient translation of LTL to limit deterministic automata, its usefulness for quantitative model checking is limited. Recently it has been shown by Sickert et al. [52] that LTL can be translated to limit deterministic

automata such that the resulting automata can be used for quantitative model checking, but the construction presented there is still double exponential for small fragments. Recall that the model checking algorithm has two parts (i) translating LTL to automata (ii) analyzing the cross product. Our results for qualitative model checking and the work by [52] are in improving the first part of the algorithm. For quantitative model checking we see how we can perform better by improving the second part. We rely on existing translations of LTL fragments to deterministic Büchi automata, and present a new algorithm for the analysis of the cross product. This new algorithm enables us to reduce the complexity of quantitative model checking for different fragments of LTL which we cover in Chapter 5.

2.2.3 Finite State Probabilistic Monitors

Courcoubetis and Yannakakis [19] showed that quantitative model checking problem for Markov Chains is PSPACE-complete, and proposed an alternate approach to solving the problem, which does not involve automata. Since Markov chains are special case of MDPs one could use deterministic automata but that would yield a double exponential time algorithm as in the case of MDPs. For Markov chains one can in fact use *unambiguous* automata [8] or *probabilistic* automata, instead of deterministic ones. Probabilistic Büchi automata (PBA) proposed in [7] are a generalization of limit deterministic Büchi automata. PBAs associate each word with a probability measure. A *cut-point*, a number in $\theta \in [0, 1]$, can be then used to define a language: the set of all words with measure $> \theta$ (or $\geq \theta$). PBAs with cut-point 1 can be used to perform quantitative verification of Markov Chains and PBAs with cut-point 0 can be used for qualitative verification. We focus on a subclass of PBAs known as Finite-state Probabilistic Monitors (FPM) [12], in which all states, except a special absorbing reject state, are accepting. FPMs have the added appeal that they can be used as runtime monitors of temporal properties.

We consider 3 special classes of FPMs in this thesis. A *strong monitor* is an FPM with cut-point 1, i.e., an input word is accepted if the probability of reaching the reject state is 0. Thus a strong monitor never rejects a good word. A *weak monitor* is an FPM with cut-point 0, i.e., it accepts a word if it has some non-zero probability of being accepted. In other words, a weak monitor never accepts a bad word. Finally a *robust monitor* is an FPM

(with cut-point θ) such that there is a gap $g > 0$ with the property that every input word is either accepted with probability at least $\theta + g$ or at most $\theta - g$. Robust monitors are automata with two-sided, bounded error. Therefore standard techniques like amplification can be used to have their error probabilities reduced. That is one can execute an input on many independent instances of the monitor and take the majority answer to have the error reduced by an exponential factor in the number of repetitions.

In Chapter 3 we investigate the construction of strong monitors, weak monitors, and robust monitors, for safety properties expressible in Linear Temporal Logic.

Chapter 3

Monitors for Safety LTL specifications

3.1 Overview

In this chapter we focus on translation of safety properties experessed in LTL to probabilistic monitors. LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$) are LTL properties built using propositions, conjunctions, disjunctions, next, and release operators; negations are restricted to being only applied to propositions (Definition 3.1). LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$) is known to capture all safety properties expressible in LTL [54].

In this chapter and sebsequent ones we will be interested in different fragments of LTL. We will use LTL($op_1, \ldots, op_k$) as the set of formulae built using literals (propositions and their negations) along with operators $op_1, \ldots, op_k$. We use $\mathcal{B}(\text{LTL}(op_1, \ldots, op_k))$ to denote all possible boolean combination of formulae in LTL($op_1, \ldots, op_k$). For example LTL($\mathbf{F}, \wedge$) represents the set of formulae built using literals and the connectives $\mathbf{F}$ and $\wedge$.

For this chapter we start out with the fragment LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$) which we will refer to as safe-LTL.

Definition 3.1 (Safe-LTL syntax). *The fragment* LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$) *over propositions AP is described by the following syntax*

$$\varphi \quad ::= \quad p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi \,\mathbf{R}\, \varphi \qquad\qquad p \in AP$$

We show that for any property φ in LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$), there is a strong monitor with $O(2^{|\varphi|})$ states that accepts the models of φ. The monitor is essentially the nondeterministic Büchi automata for the property constructed in [37], where the nondeterministic choices are turned into probabilistic choices by assigning some non-zero probability to each choice.

While this construction is not novel, it can be used to verify if a Markov chain violates a safety property with non-zero probability (qualitative model checking of co-safety) — since the PBA obtained by flipping the accept states and reject state of a FPM accepts the complement of the safety property with non-zero probability. We also prove that there is no translation from LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$) to small weak monitors. More specifically, we show that there is a family of LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$) properties $\{\varphi_n\}_{n\in\mathbb{N}}$ such that $|\varphi_n| = O(n \log n)$ and the smallest weak monitor for φ_n is of size $2^{2^{|\varphi|}}$. Since all safety properties in LTL can be recognized by a doubly exponential deterministic automata, these results show that weak monitors maybe no smaller than deterministic automata. These results are surprising in the following light. Strong monitors are known to recognize only regular safety properties [12]. On the other hand, weak monitors are known to recognize all regular persistence properties [1] and even some non-regular properties [12]. Thus, while weak monitors recognize a richer class of properties than strong monitors, they are not as efficient.

Next, we consider a fragment of LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$) denoted LTL($\mathbf{G}, \mathbf{X}, \vee, \wedge$), where we only allow the use of the "always" modal operator instead of release.

Definition 3.2. *The logic* LTL($\mathbf{G}, \mathbf{X}, \vee, \wedge$) *over set of propositions AP is described by the following syntax:*

$$\varphi \quad ::= \quad p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \mathbf{G}\varphi \qquad\qquad p \in AP$$

Since $\mathbf{G}\varphi$ *can be interpreted as* $\mathbf{false}\,\mathbf{R}\,\varphi$, LTL($\mathbf{G}, \mathbf{X}, \vee, \wedge$) *is a fragment of* LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$).

For this logic, we show that every formula φ has a weak monitor of size $O(2^{|\varphi|})$. Given results in [3], our construction demonstrates that weak monitors can be exponentially more succinct than deterministic automata for a large, natural class of properties. We also consider the construction of robust monitors for LTL($\mathbf{G}, \mathbf{X}, \vee, \wedge$). We show that for any property φ, there is a robust monitor of size $O(2^{|\varphi|})$ with gap $2^{-|\varphi|}$. Our construction is optimal in terms of the gap; we show that any robust monitor with gap $2^{-o(|\varphi|)}$ must be of size at least $2^{2^{\Omega(|\varphi|)}}$. Thus, robust monitors with subexponential gaps are no more efficient than deterministic automata for this logic. Our results are summarized in Figure 3.1.

[1] Informally, persistence properties are those that say that "eventually something always happens". For a formal definition see [41].

	DBA	**NBA**	**Strong Monitors**	**Weak Monitors**	**Robust Monitors**
LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$)	2-EXP	EXP	EXP	2-EXP	with gap $\frac{1}{2^{o(n)}}$: 2-EXP
LTL($\mathbf{G}, \mathbf{X}, \vee, \wedge$)	2-EXP	EXP	EXP	EXP	with gap $\frac{1}{2^{o(n)}}$: 2-EXP with gap $\frac{1}{2^n}$: EXP

Table 3.1: Size of automata recognizing Safety Properties in LTL. EXP means the number of states is exponential in the size of the formula and 2-EXP means that the number of states is doubly exponential. DBA stands for Deterministic Büchi Automata and NBA stands for Nondeterministic Büchi Automata.

We conclude the overview with a brief discussion of our lower bound proofs which draw on results in communication complexity [61, 38]. More specifically, we focus on *one round* protocols, where the only one message is sent from Alice to Bob, and Bob computes the value of the function based on this message and his input. We consider the *non-membership* problem, which is a special case of set disjointness problem, where Alice gets a subset $X \subseteq S$, and Bob gets $y \in S$, and they are required to determine if $y \in X$. We observe that the non-membership problem has a high VC-dimension, and, therefore, using results in [34], has a high communication complexity. Next, we observe that for certain languages, an FPM can be used to construct a one round protocol for this problem. In this protocol Alice and Bob construct special strings based on their inputs, Alice sends the state of the FPM after reading her input, and Bob computes the answer to the non-membership problem based on whether his string is accepted from the state sent by Alice. Thus, a lower bound on the communication complexity of the non-membership problem is lifted to obtain a lower bound on the number of states in an FPM recognizing certain languages.

3.2 Preliminaries

3.2.1 Monitors

We first look at the formal definition of a finite state probabilistic monitor (FPM). Recall that a stochastic matrix is a square matrix with entries in $[0, 1]$ such that every row sums up to exactly 1.

Definition 3.3 (FPMs). *A finite state probabilistic monitor* (FPM) $\mathcal{M}$ *is a tuple* $(Q, \Sigma, \mu_0, q_r, (\delta_\sigma)_{\sigma\in\Sigma})$ *where* Q *is a finite set of states;* $\mu_0 \in dist(Q)$ *is the initial distribution;* $q_r \in Q$ *is the reject state, and* $(\delta_\sigma)_{\sigma\in\Sigma}$ *is an indexed set of stochastic matrices with dimension* $|Q| \times |Q|$ *such that* $\delta_\sigma(q_r, q_r) = 1$ *for all* $\sigma \in \Sigma$.

We will use $|\mathcal{M}|$ *to denote the size of* Q. *For* $q \in Q$ *we will use* $(\mathcal{M}, q)$ *to denote the* FPM $(Q, \Sigma, \mu'_0, q_r, (\delta_\sigma)_{\sigma\in\Sigma})$ *where* $\mu'_0(q) = 1$.

Given an infinite string as input a FPM $\mathcal{M}$ behaves as follows. $\mathcal{M}$ first chooses a state q_1 according to μ_0. When the computation is at state q and the next symbol is σ then it moves to state q' with probability $\delta_\sigma(q, q')$; consumes the input symbol σ and keeps repeating this process for the remaining input.

For a finite word $u = \sigma_0 \dots \sigma_n$ define δ_u as the matrix product $\delta_{\sigma_0} \dots \delta_{\sigma_n}$, and let $\mu_{\mathcal{M},u}$ be the distribution $\mu_0\delta_u$. Define the rejection probability of u as $\mu^{rej}_{\mathcal{M},u} = \mu_{\mathcal{M},u}(q_r)$ and and acceptance probability $\mu^{acc}_{\mathcal{M},u} = 1 - \mu^{rej}_{\mathcal{M},u}$. Next observe that $\mu^{rej}_{\mathcal{M},\alpha(i)} \leq \mu^{rej}_{\mathcal{M},\alpha(i+1)}$ because the reject state has no edges leaving it. So, the sequence $\mu^{rej}_{\mathcal{M},\alpha(0)}\mu^{rej}_{\mathcal{M},\alpha(1)} \dots$ is non decreasing and since it is upper-bounded by 1, its limit exists. Define $\mathcal{M}$'s *probability of rejecting* α, denoted by $\mu^{rej}_{\mathcal{M},\alpha}$, to be this limit. Let $\mathcal{M}$'s *probability of accepting* α be $\mu^{acc}_{\mathcal{M},\alpha} = 1 - \mu^{rej}_{\mathcal{M},\alpha}$.

Given a cutpoint $\lambda \in [0, 1]$, the language recognized by $\mathcal{M}$ can be defined as the set of all infinite words with probability of acceptance at least λ or strictly more than λ. This is formally defined below.

Definition 3.4. *Given a cut-point* $\lambda \in [0, 1]$ *and FPM* $\mathcal{M}$

- $L_{>\lambda}(\mathcal{M}) = \{\alpha \in \Sigma^\omega \mid \mu^{acc}_{\mathcal{M},\alpha} > \lambda\}$

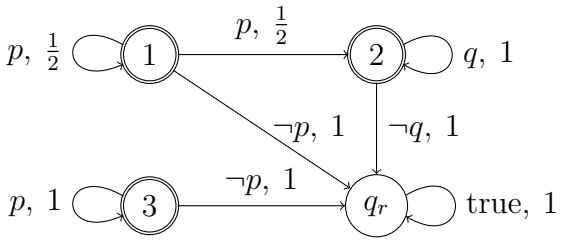


Figure 3.1: Example FPM $\mathcal{M}$

- $L_{\geq\lambda}(\mathcal{M}) = \{\alpha \in \Sigma^\omega \mid \mu^{acc}_{\mathcal{M},\alpha} \geq \lambda\}$

Example: Figure 3.1 shows an example of an FPM. Here the input alphabet consists of Boolean assignments for p and q that is $\Sigma = 2^{\{p,q\}}$. The transitions are shown to be annotated with predicates over p and q: for example the transition from 1 to 2 is annotated with $p, \frac{1}{2}$ which means if you are in state 1 and see an input symbol σ for which $p \in \sigma$ then you move to state 2 with probability $\frac{1}{2}$. Consider the initial distribution to be the one which assigns equal probabilities to states $\{1, 2, 3\}$ and let q_r be the reject state. The language associated with $\mathcal{M}$ at cutpoints 1 and 0 are: $L_{\geq 1}(\mathcal{M}) = [\![\mathbf{G}(p \wedge q)]\!]$ and $L_{>0} = [\![\mathbf{G}(p \vee \mathbf{G}q)]\!]$.

A cut-point is said to be extremal if it is either 0 or 1, and it is called non-extremal otherwise. The following proposition states that when the cutpoint is non-extermal it does not matter as to what that exact cutpoint is. The Proposition is proved in [12] and a proof sketch is included here because the same construction will be used in Corollary 3.2.

Proposition 3.1 ([12]). *For any $p \in [0, 1]$ and for any* FPM $\mathcal{M}$

- *there is an* FPM $\mathcal{M}'$ *of size $O(|\mathcal{M}|)$ such that $\mu^{rej}_{\mathcal{M}',\alpha} = p\mu^{rej}_{\mathcal{M},\alpha}$*
- *there is an* FPM $\mathcal{M}'$ *of size $O(|\mathcal{M}|)$ such that $\mu^{acc}_{\mathcal{M}',\alpha} = p\mu^{acc}_{\mathcal{M},\alpha}$.*

Proof. Let $\mathcal{M}$ be $(Q, \Sigma, \mu_0, q_r, (\delta_\sigma)_{\sigma\in\Sigma})$. First let us consider when there is a single state q_0 for which $\mu_0(q_0) = 1$. In this case we introduce two new states $q'_0, q_{acc} (\notin Q)$ and extend the transition matrices to include $\delta_\sigma(q'_0, q) := p\delta_\sigma(q_0, q)$, $\delta_\sigma(q'_0, q_{acc}) := (1 - p)$, $\delta_\sigma(q_{acc}, q_{acc}) := 1$ and $\delta_\sigma(q, q'_0) := \delta_\sigma(q'_0, q'_0) = 0$ for all $q \in Q$ and $\sigma \in \Sigma$. Let $\mathcal{M}'$ be $(Q \cup \{q'_0, q_{acc}\}, \Sigma, \mu'_0, q_r, (\delta_\sigma)_{\sigma\in\Sigma})$ where $\mu'_0(q'_0) := 1$. The construction ensures that every word is diverted to q_{acc} right at the beginning with probability $(1-p)$. A word α gets rejected on $\mathcal{M}'$ if it does not go to q_{acc} and

ends up in q_r. If it does not go to q_{acc} then the $\mathcal{M}$ would behave exactly as it would on w except for the beginning, where the probability gets scaled by p, and hence $\mu^{rej}_{\mathcal{M}',\alpha} = p\mu^{rej}_{\mathcal{M},\alpha}$. When there are multiple states $q \in Q$ with $\mu_0(q) > 0$ one can introduce a copy for each of them in the same way.

The second part can be similarly proved by diverting probabilities to q_r instead of q_{acc} at the beginning. □

Corollary 3.1. *For any two cutpoints $\lambda_1, \lambda_2 \in (0,1)$ and* FPM $\mathcal{M}_1$*, there is a* FPM $\mathcal{M}_2$ *such that $L_{>\lambda_1}(\mathcal{M}_1) = L_{>\lambda_2}(\mathcal{M}_2)$ and $L_{\geq\lambda_1}(\mathcal{M}_1) = L_{\geq\lambda_2}(\mathcal{M}_2)$ and $\mathcal{M}_2$ is of size $O(|\mathcal{M}_1|)$.*

We will be interested in FPMs for which the probability of accepting any word is bounded away from the cut-point. We call these FPMs to be robust, and we define this formally. Robustness, first introduced in [12], is analogous to the concept of isolated cut-points [48] for probabilistic automata over finite words.

Definition 3.5. *Given* FPM $\mathcal{M}$ *and a cut-point λ define $gap(\mathcal{M}, \lambda)$ as*

$$gap(\mathcal{M}, \lambda) = \inf_{\alpha \in \Sigma^\omega} |\mu^{acc}_{\mathcal{M},\alpha} - \lambda|$$

Definition 3.6. *A* FPM *is said to be* ***robust*** *with respect to a cut-point λ if $gap(\mathcal{M}, \lambda) > 0$. When a $\mathcal{M}$ is robust for cut-point λ then $L_{>\lambda}(\mathcal{M}) = L_{\geq\lambda}(\mathcal{M})$ which we will represent by $L_\lambda(\mathcal{M})$.*

For robust FPMs, we can always consider the cut-point to be $\frac{1}{2}$ without seriously changing the size of the automata or its gap. Thus in the rest of the paper, we will assume that the cut-point of any robust monitor that we consider is $\frac{1}{2}$.

Corollary 3.2. *If* FPM $\mathcal{M}$ *is robust at λ then there is a* FPM $\mathcal{M}'$ *of the same size as $\mathcal{M}$ such that $L_\lambda(\mathcal{M}) = L_{\frac{1}{2}}(\mathcal{M}')$ and $gap(\mathcal{M}', \frac{1}{2}) \geq \frac{1}{2} gap(\mathcal{M}, \lambda)$*

Proof. The construction of Proposition 3.1 give us the required FPM $\mathcal{M}'$. When $\lambda < \frac{1}{2}$ choose $p := \frac{1}{2(1-\lambda)}$ and observe that:

$$gap(\mathcal{M}', \tfrac{1}{2}) = \inf_{\alpha \in \Sigma^\omega} |\mu^{acc}_{\mathcal{M}',\alpha} - \tfrac{1}{2}| = \inf_{\alpha \in \Sigma^\omega} |\tfrac{1}{2} - \mu^{rej}_{\mathcal{M}',\alpha}|$$

$$= \inf_{\alpha \in \Sigma^\omega} |\tfrac{1}{2} - \tfrac{\mu^{rej}_{\mathcal{M},\alpha}}{2(1-\lambda)}| = \frac{\inf_{\alpha \in \Sigma^\omega} |\mu^{acc}_{\mathcal{M},\alpha} - \lambda|}{2(1-\lambda)} \geq \tfrac{1}{2} gap(\mathcal{M}, \lambda)$$

□

Monitorable languages

An FPM $\mathcal{M}$ is said to *strongly monitor* a language $L \subseteq \Sigma^\omega$ if $L = L_{\geq 1}(\mathcal{M})$, meaning the monitor $\mathcal{M}$ never declares a correct behaviour to be wrong. Similarly $\mathcal{M}$ is said to *weakly monitor* L if $L_{>0}(\mathcal{M}) = L$ which means $\mathcal{M}$ never accepts a wrong behaviour but it may occasionally reject a correct behaviour. The FPM $\mathcal{M}$ is said to *robustly monitor* L if there is a cut-point λ for which $\mathcal{M}$ is robust and $L = L_\lambda(\mathcal{M})$.

Gap amplification

Given a robust FPM $\mathcal{M}$, one can always increase the gap (and reduce the error probability) by running multiple copies of $\mathcal{M}$ in parallel. Before presenting this result, we introduce some formal definitions that will allow us to state this result precisely.

A family of FPMs is a sequence of FPMs $\{\mathcal{M}_n\}_{n\in\mathbb{N}}$. We define the size of the family as the function $s(n) = |\mathcal{M}_n|$, and the gap of the family as the function $g(n) = gap(\mathcal{M}_n, \frac{1}{2})$.

Lemma 3.1. *Let $\{\mathcal{M}_n\}$ be a family of robust* FPM*s, then there exists a family $\{\mathcal{M}'_n\}$ of size $s(n)^{\lceil \frac{1}{g(n)^2} \rceil}$ with $\Omega(1)$ gap such that for all n $L_{\frac{1}{2}}(\mathcal{M}_n) = L_{\frac{1}{2}}(\mathcal{M}'_n)$.*

Proof. Gap amplification is well known technique wherein a particular experiment is repeated in order to increase the gap or reduce the error probability [43]. Consider $\mathcal{M}'_n$ to be the machine that runs $\lceil \frac{1}{g(n)^2} \rceil$ copies of $\mathcal{M}_n$ in parallel on an input word α. $\mathcal{M}'_n$ rejects α if more than $\frac{1}{2}$ of the $\lceil \frac{1}{g(n)^2} \rceil$ copies of $\mathcal{M}_n$ reject α. Using Chernoff's bounds we can show that for any α, $|\mu^{acc}_{\mathcal{M}'_n,\alpha} - \frac{1}{2}| \geq \frac{1}{2} - e^{-2}$. The bounds on the size and the gap for $\mathcal{M}'_n$ follow from these observations. □

3.2.2 LTL to NBA

Lemma 3.2 ([59, Theorem 22, Proposition 20]). *For every formula φ in* LTL$(\mathbf{R}, \mathbf{X}, \vee, \wedge)$ *there is an NBA $N = (Q, \Sigma, Q_0, \delta, \{q\})$ such that $\delta(q, \sigma) = \{q\}$ for all $\sigma \in \Sigma$, $[\![\neg\varphi]\!] = L(N)$ and size of N is $O(2^{|\varphi|})$.*

Proof. The NBA constructed from the alternating automaton for $\neg\varphi$ has a single final state with outgoing edges only to itself. □

3.2.3 Communication Complexity

In the two-party communication complexity model of Yao [61], there are two parties Alice and Bob, who are given strings $x \in X$ and $y \in Y$, respectively, and are trying to cooperatively compute a Boolean function $f : X \times Y \to \{0, 1\}$ on their joint inputs. We are primarily interested in one-round communication protocols, wherein Alice computes some (randomized) function a on her input x, sends the output of this function to Bob, and then Bob, based on Alice's message and his own input, tries to compute the answer $f(x, y)$ using another function b. This can be formalized as below.

Definition 3.7. *A one round protocol is $P = (a, Z, b)$, where $a : X \times R_A \to Z$ is the (randomized) function that Alice computes (where R_A is the space of Alice's random choices), Z is the space of messages that Alice uses to communicate to Bob, and $b : Z \times Y \times R_B \to \{0, 1\}$ is the (randomized) function that Bob computes (where again R_B is the space of Bob's random choices).*

- *The protocol P is said to* compute f with error at most $\epsilon \in (0, 1)$ *if*

$$\Pr_{r_1 \in_R R_A, r_2 \in_R R_B} \left[b(a(x, r_1), y, r_2) = f(x, y) \right] \geq 1 - \epsilon$$

  where the probability is measured over the random choices made by Alice and Bob.

- *The* cost *of protocol P will be taken to be the number of bits communicated by Alice to Bob in the worst case, i.e., $\mathsf{cost}(P) = \log |Z|$. Notice, that in measuring the cost of the protocol, we do not measure the resources needed to compute the functions a and b.*

- *The* randomized one-round communication complexity *of function f, denoted by $R_\epsilon^{A \to B}(f)$, is the least cost of any one-round protocol computing f with error at most ϵ.*

The one-round communication compexity of a boolean function $f(x, y)$ is closely related to the concept of VC-dimension. We take a look at the definition VC-dimension [56].

Definition 3.8. *Let H be a class of boolean functions over the set Y. A set $Y' \subseteq Y$ is said to be* shattered *by H if for every $T \subseteq Y'$ there exists a function $h_T \in H$ such that for all $y \in Y'$ $h_T(y) = 1$ iff $y \in T$. The size of the largest set $Y' \subseteq Y$ which is shattered by H is known as the* VC-dimension *of H and is denoted by $VC\text{-}dim(H)$.*

For a Boolean function $f : X \times Y \to \{0,1\}$ let $f_x : Y \to \{0,1\}$ be the function defined as $f_x(y) = f(x,y)$ and let $f_X = \{f_x \mid x \in X\}$.

Lemma 3.3 ([34]). *For any boolean function $f : X \times Y \to \{0,1\}$ over finite sets X and Y, and any constant error $\epsilon < \frac{1}{8}$, the one-round randomized communication complexity of f, $R_\epsilon^{A\to B}(f) = \Omega(VC\text{-}dim(f_X))$.*

We will be interested in a particular function which we will call the *non membership function*. Given any set S the non-membership function is the Boolean function $g^S : X \times Y \to \{0,1\}$ where $X = 2^S$, $Y = S$ and $g(x,y)$ is 1 when $y \notin x$ and 0 otherwise.

Proposition 3.2. *The VC-dimension of the class of functions g_X^S is equal to the size of S, that is $VC\text{-}dim(g_X^S) = |S|$.*

Proof. For every subset $Y' \subseteq S$ the function $g_x^S \in g_X^S$ where $x = S - Y' \in X$ is such that $g_x^S(y) = 1$ iff $y \in Y'$. □

Corollary 3.3. *For error $\epsilon < \frac{1}{8}$, the one-round randomized communication complexity of the non-membership function $R_\epsilon^{A\to B}(g^S)$ is $\Omega(|S|)$*

3.3 Monitors for Safe-LTL

In this section, we present a construction of strong monitors for formulae in LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$) of exponential size, and also show that the smallest weak monitors for some formulas in LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$) can be doubly exponential sized. However, before presenting these results, we observe that any exponential blow-up is inevitable even for very simple formulas that are built using only the $\mathbf{X}$ operator. Thus the best upper bounds we can hope for is exponential.

Proposition 3.3.

- *There exists a family of specifications $\{\varphi_n\}_{n\in\mathbb{N}}$ such that any family $\{\mathcal{M}_n\}_{n\in\mathbb{N}}$ that weakly monitors it has size at least $2^{|\varphi_n|}$, and*
- *There exists a family of specfications $\{\varphi_n\}_{n\in\mathbb{N}}$ such that any family $\{\mathcal{M}_n\}_{n\in\mathbb{N}}$ that strongly monitors it has size at least $2^{|\varphi_n|}$*

Proof. Consider the class of languages $\{L_n\}$ where $L_n = \{uu(0+1)^\omega \mid u \in \{0,1\}^n\}$. L_n can be specified by saying that for each $i \in \{1,\ldots,n\}$ the ith input symbol should be the same as the $(i+n)$th input symbol, which can be done using only $\mathbf{X}$ and the boolean connectives. Any FPM that weakly recognizes L_n should have at least 2^n states because for each $u \in \{0,1\}^n$ one can identify a state q reachable from u and not reachable on any other $v \in \{0,1\}^n$ such that the word u is accepted with non-zero probability from q.

The complement of the above language, L_n, can be used to show that the translation to strong monitors will also result in an exponential blowup, and this can be argued in a similar fashion. □

3.3.1 Strong Monitors

We present our construction of an exponential size strong monitor for LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$) formulas.

Theorem 3.1. *For every formula φ in* LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$) *there is a* FPM $\mathcal{M}_\varphi$ *of size* $O(2^{|\varphi|})$ *such that $\mathcal{M}_\varphi$ strongly monitors $[\![\varphi]\!]$ that is $L_{\geq 1}(\mathcal{M}_\varphi) = [\![\varphi]\!]$*

Proof. We begin by using Lemma 3.2 to construct a NBA $B = (Q, \Sigma, Q_0, \delta, q_f)$ that recognizes all words that don't satisfy the specification φ, such that state q_f is absorbing. Let $\mu_0 \in dist(Q)$ such that $\mu_0(q) = \frac{1}{|Q_0|}$ if $q \in Q_0$ and 0 otherwise. For all $q_1, q_2 \in Q$ define $\delta_\sigma(q_1, q_2) = \frac{1}{|\delta(q_1,\sigma)|}$ if $q_2 \in \delta(q_1, \sigma)$ and 0 otherwise. The required FPM $\mathcal{M}_\varphi$ is $(Q, \Sigma, \mu_0, q_f, (\delta_\sigma)_{\sigma\in\Sigma})$. Any word that satisfies φ can never reach the reject state of the FPM, if it could then the accept state of B would also be reachable on that word and since it is absorbing the word would be accepted by B, which is not possible as B accepts words that do not satisfy φ. Hence no word satisfying φ can reach the reject state of $\mathcal{M}_\varphi$ so we have $[\![\varphi]\!] \subseteq L_{\geq 1}(\mathcal{M}_\varphi)$. Any word that does not satisfy φ can reach the accepting state of the NBA, and hence can reach the reject state in $\mathcal{M}_\varphi$ with non-zero probability and so we have that $[\![\varphi]\!] \supseteq L_{=1}(\mathcal{M}_\varphi)$. □

The above result is not a novel construction. However, it can be potentially exploited in model checking Markov Chains. Flipping the accept and reject states of the FPM gives a

PBA that accepts the complement of the safety property with non-zero probability, hence the above construction shows that there are exponential sized PBAs recognizing co-safety properties expressed in LTL. Thus, if to check if a Markov chain violates a safety property with non-zero probability, we can use this PBA instead of constructing a deterministic automaton for the co-safety property (which can be doubly exponential in size).

3.3.2 Weak Monitors

While LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$) admits strong monitors of exponential size, we show that the smallest weak monitors for some formulas can be doubly exponential in size. This is interesting in the light of the fact that weak monitors are expressively more powerful than strong monitors (i.e., can recognize languages not recognizable by strong monitors) [12].

Theorem 3.2. *There exists a family of* LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$) *specification* $\{\varphi_n\}_{n\in\mathbb{N}}$ *of size* $O(n \log n)$ *such that any family of* FPM*s that weakly monitors* $\{\varphi_n\}_{n\in\mathbb{N}}$ *has size* $2^{\Omega(2^n)}$

Proof. Consider $\Sigma = \{0, 1, \#, \$\}$ and the following ω languages over Σ

$$S_n = (\# \cdot (0+1)^n)^+ \cdot \$ \cdot (0+1)^n$$
$$R'_n = \{(\# \cdot (0+1)^n)^* \cdot (\# \cdot w) \cdot (\# \cdot (0+1)^n)^* \cdot \$ \cdot w \mid w \in (0+1)^n\}$$
$$R_n = S_n - R'_n$$
$$L_n = R_n^\omega + R_n^* \cdot (\# \cdot (0+1)^n)^\omega$$

A word in S_n can be thought of as an instance of a non-membership query: call the set of n-bit strings appearing before the \$ as the query set and a n-bit string appearing after the \$ as the query string. In a non-membership query you want to know whether the query string does not occur in the query set. R'_n represents the words in S_n that are *no* instances of the non-membership query and R_n represents the *yes* instances. L_n represents either a possibly infinite sequences of yes instances of the non-membership query or finitely many yes instances followed by an infinite sequence of n-block $0, 1$ separated by $\#$ alone. For the LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$) specification the set of propositions P we consider is $\{0, 1, \#, \$\}$, and we will assume that exactly one proposition holds at any time point. This can be easily enforced by a constant sized specification. For the sake of simplicity we present the specification

family that is of size $O(n^2)$. For details about the more succinct $O(n \log n)$ specification the interested reader should refer to [36] after reading this proof.

$$\# \wedge \mathbf{G}(\# \Rightarrow (\bigwedge_{i=1}^{n} \mathbf{X}^i(0 \vee 1)) \wedge \mathbf{X}^{n+1}(\# \vee \$)) \tag{3.1}$$

$$\wedge \mathbf{G}(\$ \Rightarrow (\bigwedge_{i=1}^{n} \mathbf{X}^i(0 \vee 1)) \wedge \mathbf{X}^{n+1}(\#)) \tag{3.2}$$

$$\wedge \mathbf{G}(\# \Rightarrow \bigvee_{i=1}^{n} \bigvee_{\sigma \in \{0,1\}} (\mathbf{X}^i(\sigma) \wedge (\mathbf{X}(\$ \wedge \mathbf{X}^i(\sigma^c)) \,\mathbf{R}\, (\neg\$)))) \tag{3.3}$$

(3.1) says that the words should begin with # and each # should be followed by a n-bit string followed by a # or \$. (3.2) says that every \$ is followed by n-bit string block then followed by #. (3.1) and (3.2) together describe a sequence of possibly infinite non-membership queries. (3.3) says that for any every word in the query set differs from the query string in at least one position. Hence (3.1) $\wedge$ (3.2) $\wedge$ (3.3) is a specification of L_n and is of size $O(n^2)$.

Now let us assume that we have a family of FPMs $\{\mathcal{M}_n\}$ that weakly monitors $\{L_n\}$. We make the following claim about $\mathcal{M}_n$.

Claim: Consider any $n \in \mathbb{N}$ and $c \in (0,1)$. Let $\mathcal{M}_n$ be $(Q, \Sigma, Q_0, q_r, (\delta_\sigma)_{\sigma \in \Sigma})$. There exists a $u \in R_n^*$ and a state $q \neq q_r$ with $\mu_{\mathcal{M}_n,u}(q) > 0$ such that for all $\beta \in R_n^\omega : \mu^{acc}_{(\mathcal{M}_n,q),\beta} \geq c$.

Proof: Suppose the claim does not hold for some c and n. Let us fix $\mathcal{M}$ to be $\mathcal{M}_n$. Since the claim is false we have that for any $u \in R_n^*$ and any q with $\mu_{\mathcal{M},u}(q) > 0$ there is a $\beta_q \in R_n^\omega$ such that the measure of accepting runs from q on β_q is less than c. Let us fix q to be a state with maximal $\mu_{\mathcal{M},u}(q)$ among all $q \neq q_r$. For such a q we have $\mu_{\mathcal{M},u}(q) \geq \frac{\mu^{acc}_{\mathcal{M},u}}{|\mathcal{M}|}$ by pigeon-hole principle. For an FPM the acceptance measure of any string is non-increasing along its length and so there should be a finite prefix of β_q, say $v \in R_n^*$ such that $\mu^{acc}_{(\mathcal{M},q),v} < c$. From this we get

$$\mu^{acc}_{\mathcal{M},uv} \leq \mu^{acc}_{\mathcal{M},u}\left(1 - \frac{1-c}{|\mathcal{M}|}\right)$$

because at least $1 - c$ of the probability of reaching q is lost to q_r after seeing v. So for any $u \in R_n^*$ we manage to find a string $v \in R_n^*$ such that the acceptance probability of the extended string uv compared to u decreases by a constant factor. But observe that uv is once again in R_n^*. So this extension process can be repeated forever to get a string in R_n^ω which is accepted with 0 probability which is a contradiction. So our claim is indeed true.

Continuing with the proof our theorem, consider $S = \{0,1\}^n$. In order to show the required lower bound on $\mathcal{M}_n$ we will show how $\mathcal{M}_n$ can be used to construct a constant-error one-round protocol for the non-membership function g^S. For the sake of the remaining argument we instantiate c in the above claim to $\frac{7}{8}$. Consider the state q as per our claim. Let η denote the string $(\#a^n)^\omega$. We make the following observations:

- for $w \in R_n$ the acceptance probability of $w\eta$ starting from q is at least $\frac{7}{8}$. This is because from q every prefix w' of $w\eta$ should be accepted with probability at least $\frac{7}{8}$, otherwise we can attach an appropriate suffix to w' to get a string that contradicts our claim.

- for $w \in R'_n$, the string $w\eta$ is accepted with 0 probability because $\mathcal{M}_n$ should accept strings $uw\eta$ with 0 probability.

So $\mathcal{M}_n$ if started from q is able to significantly distinguish between *yes* and *no* instances of the non-membership query. We can use this to construct a one-round protocol for the function g^S: Alice encodes her input set of n-bit blocks as a string in $(\#\cdot(0+1)^n)^+\$$ and runs it on $\mathcal{M}_n$ starting from q and sends to Bob the resulting state q'. Bob then simulates $y\eta$ from q' and outputs 0 iff the simulation results in rejection. (Bob cannot actually run the infinite string η but he can simulate η's acceptance because probability of η being accepted from any state can be calculated). This gives us a randomized one-round protocol with error $< \frac{1}{8}$. The number of bits exchanged in this protocol is $\log_2 |\mathcal{M}|$, but according to Corollary 3.3 any such protocol needs to exchange at least $\Omega(2^n)$ bits. Hence we get that $\mathcal{M}$ has at least $2^{\Omega(2^n)}$ states. □

3.4 Monitors for LTL($\mathbf{G}, \mathbf{X}, \vee, \wedge$)

The results in Section 3.3 show that for general LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$) formulas, weak monitors can be as large as deterministic automata. In this section, we show that when we consider the sub-logic LTL($\mathbf{G}, \mathbf{X}, \vee, \wedge$), we can demonstrate that weak monitors can be exponentially more succinct than their deterministic counterparts. The idea behind the construction is that we consider each state to represent a guess about the truth of all subformulae of the

form $\mathbf{G}\psi$: whether it holds now, or holds starting from some point in the future, or never holds. Then we argue that for satisfying behaviours one can find accepting runs that need to make finitely many correct guesses and vice versa.

In the latter part of the section we present constructions of robust monitors for this logic that are small. This result relies on considering a normalized form of the formula which yields an efficient way to construct the required monitor.

3.4.1 Weak Monitors

Before we present our construction of exponential sized weak monitors for the fragment $\mathrm{LTL}(\mathbf{G}, \mathbf{X}, \vee, \wedge)$, we introduce some assumptions and definitions that will facilitate the proof.

For a formula φ let $Sub(\varphi)$ denote the set of all subformulae of φ, and let $GSub(\varphi) \subseteq Sub(\varphi)$ be those which are of the form $\mathbf{G}\psi$.

Definition 3.9. *An **annotation** for a formula φ is a function mapping $GSub(\varphi)$ to the set $\{\top, \bot, \mathsf{L}\}$. Denote by A the set of all annotations. An annotation is called **stable** if it maps $GSub(\varphi)$ to $\{\top, \bot\}$. Given an annotation a and $\sigma \in 2^P$ an **evaluation** for φ is the unique function $e_a^\sigma : Sub(\varphi) \to \{\top, \bot\}$ that meets the following constraints:*

$$e_a^\sigma(\psi) = \top \text{ iff } (a(\psi) = \top) \text{ for } \psi \in GSub(\varphi)$$

$$e_a^\sigma(p) = \top \text{ iff } p \in \sigma \qquad e_a^\sigma(\psi_1 \wedge \psi_2) = e_a^\sigma(\psi_1) \wedge e_a^\sigma(\psi_2)$$

$$e_a^\sigma(\neg p) = \top \text{ iff } p \notin \sigma \qquad e_a^\sigma(\psi_1 \vee \psi_2) = e_a^\sigma(\psi_1) \vee e_a^\sigma(\psi_2)$$

An annotation represents a guess for each subformula $\mathbf{G}\psi$ stating whether $\mathbf{G}\psi$ holds now ($\top$), holds for some later point but not now (L), never holds ($\bot$). An evaluation attempts to evaluate the truth for all the subformulae (read $e_a^\sigma(\psi)$ as evaluation of ψ annotated with a and σ). Note that an evaluation need not be logically consistent. For example $e_a^\sigma(\mathbf{G}p)$ could be $\top$ because $a(\mathbf{G}p) = \top$, but $e_a^\sigma(p) = \bot$ because $p \notin \sigma$. We are now ready to present the main result of this section.

Theorem 3.3. *For every $\varphi \in \mathrm{LTL}(\mathbf{G}, \mathbf{X}, \vee, \wedge)$ there is a FPM $\mathcal{M}_\varphi$ of size $2^{O(|\varphi|)}$ such that $L_{>0}(\mathcal{M}_\varphi) = [\![\varphi]\!]$*

Proof. First we show how the construction works for $\varphi \in \mathrm{LTL}(\mathbf{G}, \mathbf{X}, \vee, \wedge)$ when it does not have any $\mathbf{X}$ operators. Let Q the set of states be $(A \times \{0, 1\}) \cup \{q_r\}$. For $\sigma \in 2^P$ define T_σ

to be the binary relation on Q such that for $a, b \in A$, $((a,t_1),(b,t_2)) \in T_\sigma$ iff: $t_2 = 0$ and if $t_1 = 1$ then $e_a^\sigma(\varphi) = \top$ and the following conditions on a, b holds:

$$(a(\mathbf{G}\psi) = \top) \Rightarrow (b(\mathbf{G}\psi) = \top \wedge e_a^\sigma(\psi) = \top)$$
$$(a(\mathbf{G}\psi) = \bot) \Rightarrow (b(\mathbf{G}\psi) = \bot)$$
$$(a(\mathbf{G}\psi) = \mathsf{L}) \Rightarrow (b(\mathbf{G}\psi) \neq \bot)$$

For $a \in A$, $((a,t_1),q_r) \in T_\sigma$ if for no (b,t_2) the above conditions hold. For all σ, $(q_r,q_r) \in T_\sigma$. Define δ_σ as:

$$\delta_\sigma(q_1,q_2) = \frac{T_\sigma(q_1,q_2)}{\sum_{q_2} T_\sigma(q_1,q_2)},$$

Define μ_0 as follows: for an annotation a, $\mu_0((a,t)) = \frac{1}{|A|}$ if $t = 1$ and define it to be 0 on the rest of the states. The required FPM $\mathcal{M}_\varphi$ is $(Q, 2^P, \mu_0, q_r, (\delta_\sigma)_{\sigma\in\Sigma})$. Now we prove that $L_{>0}(\mathcal{M}_\varphi) = [\![\varphi]\!]$.

$L_{>0}(\mathcal{M}_\varphi) \supseteq [\![\phi]\!]$: For a string α that satisfies φ we look at the sequence of states induced by α, i.e define the ith state (a_i,t_i) as $t_i = 1$ iff $i = 0$ and

$$a_i(G\psi) = \begin{cases} \top & \text{if } \alpha_i \vDash \mathbf{G}\psi \\ \mathsf{L} & \text{if } \alpha_i \nvDash \mathbf{G}\psi \text{ and } \exists j > i : \alpha_j \vDash \mathbf{G}\psi \\ \bot & \text{if } \forall j \geq i : \alpha_j \nvDash \mathbf{G}\psi \end{cases}$$

First let us observe that $\delta_{\alpha(i)}((a_i,t_i),(a_{i+1},t_{i+1})) > 0$ for any i: If $a_i(\mathbf{G}\psi) = \top$ then by construction, $\alpha_i \vDash \mathbf{G}\psi$. This implies $\alpha_{i+1} \vDash \mathbf{G}\psi$ and so $a_{i+1}(\mathbf{G}\psi) = \top$. Also one can prove that for all $i \in \mathbb{N}$ and $\psi \in Sub(\varphi)$, $(\alpha_i \vDash \psi) \Rightarrow e_{a_i}^{\alpha(i)}(\psi) = \top$ by induction on the structure of ψ . If $a_i(\mathbf{G}\psi) = \mathsf{L}$ then we know ψ is going to hold forever from some point after i, which means it is also going to hold forever from some point after $i+1$ and hence $a_{i+1}(\mathbf{G}\psi) \neq \bot$. If $a_i(\mathbf{G}\psi) = \bot$ then we know ψ is going to be false infinitely often from i, so ψ will be false infinitely often from $i+1$ as well hence $a_{i+1}(\mathbf{G}\psi) = \bot$. This makes sure that a_i and a_{i+1} are properly related. Since $\alpha \vDash \varphi$ we have that $e_{a_0}^{\alpha(0)}(\varphi) = 1$. Thus $((a_0,1),(a_1,0)) \in T_{\alpha(0)}$. Hence, $(a_0,1)(a_1,0)\ldots$ is a valid run of $\mathcal{M}_\varphi$ over α.

For any a_i if $\mathbf{G}\psi \in GSub(\varphi)$ is marked L then there is a $j > i$ such that $\mathbf{G}\psi \vDash \alpha_j$ (defnition of a_i), so it follows that $a_j(\mathbf{G}\psi) = \top$. This implies that every L eventually becomes $\top$, so

there exists a point k at which a_k marks all formulae in $GSub(\varphi)$ as either $\top$ or $\bot$ (which cannot be further modified), and hence $a_{k'} = a_k$ for all $k' \geq k$. Once you reach a_k you cannot go to any other annotation except itself and hence for all $k' \geq k : \delta_{\alpha(k')}((a_k, 0), (a_k, 0)) = 1$. Therefore this run does not have to make any probabilistic choice after the point k and hence has positive acceptance probability.

$L_{>0}(\mathcal{M}_\varphi) \subseteq [\![\varphi]\!]$: Consider any valid accepting run $(a_0, t_0)(a_1, t_1)\ldots$ of $\mathcal{M}_\varphi$ on input α. First observe that in any valid run of $\mathcal{M}_\varphi$ the number of $\top$s and $\bot$s are non decreasing. Since there are finitely many states, the run ultimately stagnates at a particular state. Let us denote by C_a the set of runs which stay in the state $(a, 0)$ after finitely many steps. Denote by C_a^i the set of all runs in C_a that stay in a after i steps. We have $C_a = \cup_{i=1}^{\infty} C_a^i$.

Fix a to be an unstable annotation. For any σ we have $\delta_\sigma((a, 0), (a, 0)) < 1$, because a has a choice to change a L to $\top$ and move to a different annotation. So the probabiltiy measure associated with C_a^i is 0 because after i steps the only transition taken is from $(a, 0)$ to $(a, 0)$ which leaks at least $1 - \delta_\sigma((a, 0), (a, 0))$ probability out of $(a, 0)$. This implies the probability associated with the set C_a is also zero.

So if the set of all accepting runs of α has non-zero measure then it has to have a run that ultimately reaches a stable annotation. Now with such a run we prove that the word α satisfies φ. For any $\varphi' \in Sub(\varphi)$ and $i \in \mathbb{N}$ if $e_{a_i}^{\alpha(i)}(\varphi') = \top$ then $\alpha_i \vDash \varphi'$, this can be proved by performing induction on the structure of φ'. The interesting case is when $\varphi' = \mathbf{G}\psi$. The definition of e suggests that if $e_{a_i}^{\alpha(i)}(\mathbf{G}\psi) = \top$ then $a_i(\mathbf{G}\psi) = \top$. Since $((a_i, t_i), (a_{i+1}, t_{i+1})) \in T_{\alpha(i)}$, we get from the definition of T that $e_{a_i}^{\alpha(i)}(\psi) = \top$ and so it follows that $\alpha_i \vDash \psi$ from the induction hypothesis. But if $\mathbf{G}\psi$ is marked $\top$ in a_i then it is marked $\top$ in every a_j for $j > i$. So $\alpha_j \vDash \psi$ for every $j \geq i$ and hence we have that $\alpha_i \vDash \mathbf{G}\psi$. Finally observe that $((a_0, t_0), (a_1, t_1)) \in T_{\alpha(0)}$ iff $e_{a_0}^{\alpha(0)}(\varphi) = \top$. Thus $\alpha_0 \vDash \varphi$.

What remains is to be shown is the construction in the presence of $\mathbf{X}$ operators. First we push down the $\mathbf{X}$ operators to the bottom, this is possible because $\mathbf{X}$ distributes over all other operators. If the number of nested $\mathbf{X}$s is at most k (which is at most $|\varphi|$) then by looking ahead k positions into the input one can evaluate the $\mathbf{X}$ subformulae just like literals. So by maintaining the last k input symbols and delaying the computation by those many steps will give use the required construction. Since we need to remember k input symbols

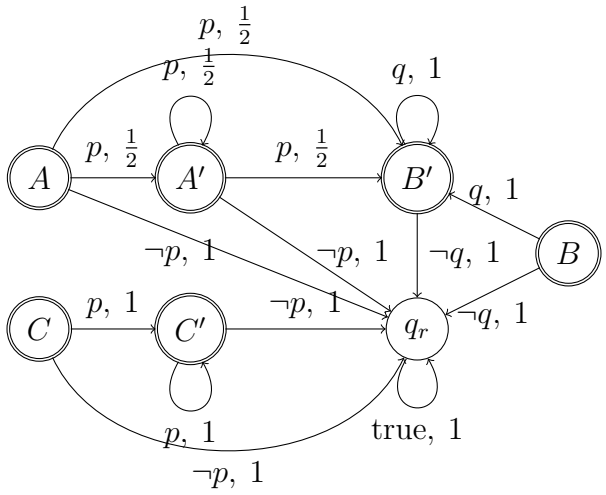


Figure 3.2: Weak monitor for $\mathbf{G}(p \vee \mathbf{G}(q))$

the construction will blow up only by a factor of $|\Sigma|^k$ which is again in $2^{|\varphi|}$. □

3.4.2 Robust Monitors

We will present a construction of robust automata of exponential size for formulae in LTL($\mathbf{G}, \mathbf{X}, \vee, \wedge$). We begin by observing that the above construction for weak monitors does not result in a robust monitor. Consider the formula $\mathbf{G}(p \vee \mathbf{G}q)$. The construction of Theorem 3.3 results in the FPM given in Figure 3.2. The initial distribution gives equal probability to the states A, B and C. The states are as follows: $A = (a_1, 1)$, $B = (a_2, 1)$, $C = (a_3, 1)$, $A' = (a_1, 0)$, $B' = (a_2, 0)$ and $C' = (a_3, 0)$, where a_1, a_2, a_3 are annotations as given below.

$$\{a_1(\mathbf{G}(p \vee \mathbf{G}q)) = \top, a_1(\mathbf{G}q) = \mathsf{L}\} \quad \{a_2(\mathbf{G}(p \vee \mathbf{G}q)) = \top, a_2(\mathbf{G}q) = \top\}$$
$$\{a_3(\mathbf{G}(p \vee \mathbf{G}q)) = \top, a_3(\mathbf{G}q) = \bot\} \tag{3.4}$$

The rest of the annotations do not appear as they are unreachable. To see why the FPM is not robust we consider the word $p^n q^\omega$. After seeing the first $n > 0$ input symbols of this word, the monitor is going to be in state A' with probabitlity $\frac{1}{3.2^n}$, in state B' with $\frac{1}{3.2^n}$ and in state C' with probability $\frac{1}{3}$. Probability of being in state C' goes to 0 as we see the rest of string q^ω. This means as n grows larger the word $p^n q^\omega$ is accepted with negligible probability

and hence the language $[\![\mathbf{G}(p \vee \mathbf{G}q)]\!]$ is not robustly monitored by this FPM. Therefore, we present a new construction that avoids these pitfalls.

Theorem 3.4. *For every φ in* LTL$(\mathbf{G}, \mathbf{X}, \vee, \wedge)$ *there is a* FPM $\mathcal{M}_\varphi$*, which is robust with* $\frac{1}{2^{|\varphi|}}$ *gap,* $2^{O(|\varphi|)}$ *states such that $\mathcal{M}_\varphi$ recognizes φ.*

Proof. As in Theorem 3.3 we can push all the $\mathbf{X}$ in the formula to the bottom and take care of it by remembering the last k input symbols where k is the nesting depth of the $\mathbf{X}$s. Therefore we are going to consider only formulae without any $\mathbf{X}$ in this proof.

The FPM that we construct is going to accept safe inputs with probability 1 and reject bad inputs with probability $> \frac{1}{2^{|\varphi|}}$.

Let us first consider the simpler logic LTL$(\vee, \mathbf{G})$ built using literals, disjunction and $\mathbf{G}$; so the formulas have no conjunction. We will say that a formula $\varphi \in$ LTL$(\vee, \mathbf{G})$ is *guarded* iff every subformula $\mathbf{G}\psi$ of φ is of the form $\mathbf{G}(\alpha \vee \beta)$, where α is a disjunction of literals, and β is a disjunction of formulae like $\mathbf{G}\gamma$; β could be an empty disjunction. Observe that every formula φ in LTL$(\vee, \mathbf{G})$ is equivalent to a guarded formula ψ such that $|\psi| = O(|\varphi|)$. This is because we have $\mathbf{G}\mathbf{G}\psi \equiv \mathbf{G}\psi$, and $\mathbf{G}(\mathbf{G}\alpha_1 \vee \mathbf{G}\alpha_2 \vee \cdots \vee \mathbf{G}\alpha_n) \equiv (\mathbf{G}\alpha_1 \vee \mathbf{G}\alpha_2 \vee \cdots \vee \mathbf{G}\alpha_n)$. Every guarded formula φ can be recognized by a deterministic monitor [2] of size $2^{|\varphi|}$, whose states keep track of the guarded subformulae which are yet to be violated. For example, if the formula is $\mathbf{G}(\alpha_1 \vee \mathbf{G}\alpha_2 \vee \mathbf{G}\alpha_3)$ then the automaton monitors α_1 until it becomes false and then starts monitoring $\mathbf{G}\alpha_2$ and $\mathbf{G}\alpha_3$ (which are guarded).

Consider $\varphi \in$ LTL$(\mathbf{G}, \mathbf{X}, \vee, \wedge)$. Since $\wedge$ distributes over $\vee$ and $\mathbf{G}(\psi_1 \wedge \psi_2) \equiv (\mathbf{G}\psi_1) \wedge (\mathbf{G}\psi_2)$, we can pull all the conjunctions out, and show that φ is equivalent to a formula ψ which is conjunction of formulas in LTL$(\vee, \mathbf{G})$. We can also see that $|\psi| = O(2^{|\varphi|})$. The FPM for ψ will be will be a disjoint union of the deterministic monitors recognizing each of the conjuncts in ψ. Thus, the number of states in this FPM is thus $O(2^{2|\varphi|})$. The initial distribution assigns equal probability to the initial states of each of the deterministic monitors (and 0 probability to all other states). A bad input violates one of the conjuncts, and so the monitor corresponding to that conjunct will reject the input. Thus, bad inputs are accepted with probability at most $(1 - \frac{1}{2^{|\varphi|}})$. On the other hand, a good input is accepted

[2] A deterministic monitor is an FPM in which each transition matrix has entries which are either 0 or 1.

by the monitors of each of the conjuncts, and therefore accepted by the FPM for ψ with probability 1. □

The proof of Theorem 3.4 constructs a robust FPM with exponential gap. We show that these bounds on the gap cannot be improved without increasing the number of states to doubly exponential; this is the content of the next theorem.

Theorem 3.5. *There exists a family of* LTL($\mathbf{G}, \mathbf{X}, \vee, \wedge$) *(and hence* LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$)*) specifications* $\{\varphi_n\}_{n\in\mathbb{N}}$ *of size* $O(n)$ *s.t. any family of robust* FPM*s with* $\frac{1}{2^{o(n)}}$ *gap that recognizes it has size* $2^{2^{\Omega(n)}}$.

Proof. We consider the specfications used in [3] to prove lower bounds on deterministic generators. Let $\varphi_n = \mathbf{G}(\bigvee_{i=1}^{n}(\neg p_i \wedge \mathbf{G}\neg q_i))$. We will reduce the problem of finding efficient protocols for the non-membership function to the problem of finding small sized FPMs for this specification.

Let $P_p = \{p_1, \ldots, p_n\}$, $P_q = \{q_1, \ldots, q_n\}$, Σ_k be $\{a \subseteq P_p \mid |a| = k\}$ and Γ_k be $\{b \subseteq P_q \mid |b| = k\}$. We choose k to be $\frac{n}{2}$ so that $|\Sigma_k| = 2^{\Omega(n)}$. For any $\sigma \subseteq P_p$ let $q(\sigma) = \{q_i \mid p_i \notin \sigma\}$. Let $S = \Sigma_k$ and g^S be the corresponding non-membership function. Using Corollary 3.3 we get that for $\epsilon \leq \frac{1}{8}$ the communication complexity $R_\epsilon^{A\to B}(g^S) \in 2^{\Omega(n)}$.

We begin by showing that a constant gap family $\{\mathcal{M}_n\}$ recognizing $\{\varphi_n\}$ should have large size. Consider an FPM family $\mathcal{M}_n$ with gap at least $\frac{3}{8}$ such that $L(\mathcal{M}_n) = [\![\varphi_n]\!]$. Now we are going construct a randomized one-round protocol for g^S with $< \frac{1}{8}$ error using $\mathcal{M}_n$. Alice encodes her input x as a string $\sigma_1\sigma_2\ldots\sigma_m$, an enumeration of the sets in x (which are also symbols in the alphabet $2^{P_p\cup P_q}$), runs it on $\mathcal{M}_n$ and gives the resulting state to Bob. Bob whose input is y simulates the word $q(y)(\emptyset)^\omega$ from the given state, and outputs 0 if it results in rejection and outputs 1 otherwise. A word violates φ_n iff there is a point in the word where for each p_i that is false it is the case that eventually q_i is true. Suppose x and y are such that $g^S(x, y) = 0$, this means that there is some $\sigma_j \in x$ for which $y = \sigma_j$ and so we get that $\sigma_1\sigma_2\ldots\sigma_m q(y)(\emptyset)^\omega$ violates φ_n. Similarly if $g^S(x, y) = 1$ then is no j such that $\sigma_j \in x$ and $y = \sigma_j$ and so $\sigma_1\sigma_2\ldots\sigma_m q(y)(\emptyset)^\omega$ satisfies φ_n. Since $\mathcal{M}_n$ has a gap of $\frac{3}{8}$ it follows that the protocol that we constructed has an error of at most $\frac{1}{8}$ in deciding the output $g^S(x, y)$. The number of bits exchanged in this protocol is $\log_2 |\mathcal{M}_n|$. But above we

saw that any such protocol should exchange $2^{\Omega(n)}$ bits which imples that the size of $\mathcal{M}_n$ has to be $2^{2^{\Omega(n)}}$. Having shown that constant gap FPMs recognizing $\{\varphi_n\}$ is of size $2^{2^{\Omega(n)}}$ we invoke Lemma 3.1 to get the same lower bound for $\frac{1}{2^{o(n)}}$ gap FPMs. □

3.5 Conclusion

In this chapter we gave constructions of FPMs for safety properties expressed in LTL. We showed that LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$) has strong monitors of exponential size, where as weak monitors and robust monitors with sub-exponential gaps, can be doubly exponentially large. For the sub-logic LTL($\mathbf{G}, \mathbf{X}, \vee, \wedge$) we gave constructions of weak monitors and robust monitors of exponential size. However, the gap for robust monitors for LTL($\mathbf{G}, \mathbf{X}, \vee, \wedge$) given by our construction is exponential and we showed that these bounds on the gap cannot be improved without increasing the number of states to doubly exponential.

A number of questions remain open. While we showed that robust monitor with sub-exponential gap for LTL($\mathbf{R}, \mathbf{X}, \vee, \wedge$) can be doubly exponential in size, we could not conclude if the construction can be improved if we relax the bounds on the gap. In particular, it would be interesting to know if there are exponential sized robust monitors with an exponential gap.

Chapter 4

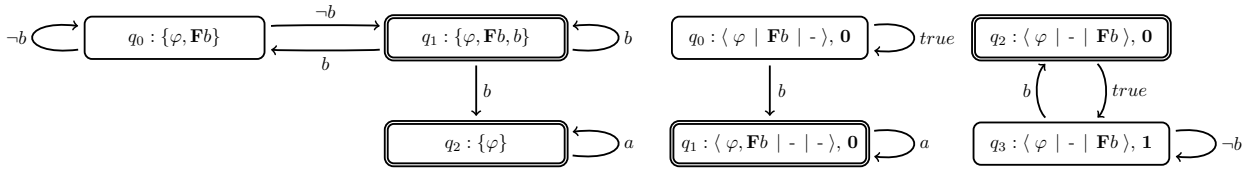
Qualitative Model Checking

In this chapter we present the construction of limit deterministic Büchi automata (LDBA) for LTL which is exponential sized for the fragment LTL_{D}. We present our construction in steps. We first look at some of the core ideas involved by looking at the translation for the subfragment $\text{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$ in Section 4.1 and then present the full construction in Section 4.2. We show that our construction is optimal for the fragment LTL_{D} in Section 4.3, and discuss the expressive power of LTL_{D} in Section 4.4. The resulting complexity improvements of qualitative model checking are covered in Section 4.5.

4.1 LDBA construction for $\text{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$

Recall that the fragment $\text{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$ consists boolean combinations of LTL properites built using literals, conjuctions, disjunctions, always, and eventually operators.

First, let us look at an example that shows that the standard construction (Fischer-Ladner and its variants) is not limit deterministic. The standard construction involves guessing the set of subformulae that are true at each step and ensuring the guess is correct. For $\varphi = \mathbf{G}(a \vee \mathbf{F}b)$ this gives us the automaton (after pruning unreachable states and merging



(a) Standard Construction

(b) Tripartition Construction

Figure 4.1: Automata for $\mathbf{G}(a \vee \mathbf{F}b)$

bisimilar ones. Here all 3 states are initial) in Figure 4.1a which is not limit deterministic as the final state q_1 has non-deterministic choices enabled.

The following proposition embodies the key idea behind our construction. A proof is provided in Appendix A.

Proposition 4.1 (Key Idea). *For any formula $\varphi \in$ LTL over P, and any word $w \in (2^P)^\omega$ exactly one of the following three holds*

$$w \vDash \neg\mathbf{F}\varphi, \quad w \vDash (\neg\mathbf{G}\varphi \wedge \mathbf{F}\varphi), \quad w \vDash \mathbf{G}\varphi$$

Furthermore, if φ is of the form $\mathbf{F}\psi$ or $\mathbf{G}\psi$ then we can deduce if $w \vDash \varphi$ holds from knowing which one of the above three holds.

Now we see how this key idea is used. Given an LTL($\mathbf{F}, \mathbf{G}, \wedge, \vee$) formula, for each of its $\mathbf{G}$-subformula we are going to predict whether it is: always true (α), true at some point but not always (β), never true (γ). Note that for any formula if we predict α/γ then the prediction should remain the same going forward. For a $\mathbf{G}$-subformula, $\mathbf{G}\psi$, if we predict β it means we are asserting $\mathbf{FG}\psi \wedge \neg\mathbf{G}\psi$ and therefore the prediction should remain β until a certain point and then change to α. This prediction entails two kinds of non-deterministic choices: **(i)** the initial choice of assigning one of α, β, γ **(ii)** if assigned β initially then the choice of the time point at which to change it to α. The first choice needs to be made once at the beginning and the second choice has to be made eventually in a finite time. They together only constitute finitely many choices which is the source of the limit determinism. We similarly define predictions for $\mathbf{F}$-subformulae as: never true (α), true at some point but not always (β), always true (γ). Notice that we have flipped the meaning of α and γ here, this is to ensure β becomes α eventually as for $\mathbf{G}$-subformulae. An *FG-prediction* for a formula $\varphi \in$ LTL($\mathbf{F}, \mathbf{G}, \wedge, \vee$), denoted by π, is a tri-partition $\langle \alpha(\pi), \beta(\pi), \gamma(\pi) \rangle$ of its $\mathbf{F}, \mathbf{G}$-subformulae. We drop π when it is clear from the context. The *prediction* for a subformula ψ made by π is said to be $\alpha/\beta/\gamma$ depending upon the partition of π in which ψ is present. The space of all FG-predictions for φ is denoted by $\Pi(\varphi)$. Table 4.1 summarizes how we interpret a FG-prediction as one of three guesses for $\mathbf{F}$ and $\mathbf{G}$ subformulae.

Example 4.1. *Consider the formula $\varphi = \mathbf{G}(a \vee \mathbf{F}b)$, and an FG-prediction $\pi = \langle \alpha, \beta, \gamma \rangle$ for φ where $\alpha = \{\varphi\}$, $\beta = \{\mathbf{F}b\}$ and $\gamma = \emptyset$. For the formula φ the prediction made is α.*

	$\alpha(\pi)$	$\beta(\pi)$	$\gamma(\pi)$
$\mathbf{F}\psi$	$\neg\mathbf{F}\psi$	$\mathbf{F}\psi \wedge \neg\mathbf{GF}\psi$	$\mathbf{GF}\psi$
$\mathbf{G}\psi$	$\mathbf{G}\psi$	$\neg\mathbf{G}\psi \wedge \mathbf{FG}\psi$	$\neg\mathbf{FG}\psi$

Table 4.1: Guess corresponding to a tripartition π

Since it is a $\mathbf{G}$*-formula this prediction says that* φ *is always true or simply* φ *is true. For the subformula* $\mathbf{F}b$ *the prediction made is* β*. This prediction says that* $\mathbf{F}b$ *is true at some point but not always which implies* $\mathbf{F}b$ *is true but not* $\mathbf{GF}b$*.*

The automaton for LTL($\mathbf{F}, \mathbf{G}, \wedge, \vee$) essentially makes a non-deterministic choice for π initially and at each step makes a choice of whether to move some formula(e) from β to α. The correctness of predictions made by π is monitored inductively. Suppose our prediction for a formula $\mathbf{G}\psi$ is α at some instant: this implies we need to check that ψ is true at every time point there onwards (or equivalently check that ψ is true whenever α is predicted for $\mathbf{G}\psi$ since the prediction α never changes). If we are able to monitor the ***truth*** of ψ at every instant then it is clear how this can be used to monitor the ***prediction*** α for $\mathbf{G}\psi$. The crucial observation here is that the correct prediction for $\mathbf{G}/\mathbf{F}$ formula gives us their truth (Proposition 4.1): a $\mathbf{G}/\mathbf{F}$ formula is true/false (respectively) at a time point if and only if its correct prediction is α at that time. Now the prediction α for $\mathbf{G}\psi$ can be checked by using the truths (derived from the predictions) of the subformulae of ψ (inductive step). If ψ is propositional then its truth is readily available from the input symbol being seen (base case of the induction). This inductive idea shall be used for all predictions. Note that since our formulae are in negation normal form we only need to verify a prediction is correct if it asserts the truth rather than falsehood of a subformula. Therefore the predictions β, γ for $\mathbf{G}\psi$ need not be checked. In case of $\mathbf{F}\psi$ the prediction α need not be checked (as it entails falsehood of $\mathbf{F}\psi$) but β, γ do need to be checked. If our prediction for $\mathbf{F}\psi$ is β then we are asserting ψ is true until a certain point in the future at which the prediction becomes α. Therefore we only need to check that ψ is true when the prediction for $\mathbf{F}\psi$ changes to α. Once again we can inductively obtain the truth of ψ at that instant from the predictions for

the subformulae of ψ and from the next input. With these two predictions in mind (α for $\mathbf{G}\psi$ and $\beta \to \alpha$ for $\mathbf{F}\psi$) we define the following set of formulae:

$$\Psi(\pi_1, \pi_2) = \{\psi \mid \mathbf{F}\psi \in \beta(\pi_1) \cap \alpha(\pi_2) \text{ or } \mathbf{G}\psi \in \alpha(\pi_1)\} \tag{4.1}$$

For checking a prediction γ about $\mathbf{F}\psi$ we need to check ψ is true infinitely often. For this purpose we use the Büchi acceptance where the final states are those where ψ is evaluated to be true, again inductively. When we are monitoring multiple $\mathbf{F}\psi$ for γ we will need a counter to cycle through all the $\mathbf{F}\psi$ in γ. Let m be the number of $\mathbf{F}\psi$ in γ. Observe that the set of formulae predicted to be γ never changes once fixed at the beginning and hence m is well defined once γ is fixed at the beginning. If the counter has value n and the ψ corresponding to the n^{th} $\mathbf{F}\psi \in \gamma$ evaluates to true , then the counter is incremented cyclically to $n+1 (\text{mod } m)$, otherwise it remains the same. The initial states are those in which the top formula evaluates to true given the predictions in that state and the input symbol to be seen. Since the input symbol is not available in the states we choose to define the initial condition over the transitions, which does not change the expressive power of the automata. The final states are those where no formula is assigned β and the counter is 0. Summarizing, a state in our automata has two components: **(a)** an FG-prediction $\pi = \langle \alpha, \beta, \gamma \rangle$ (a tri-partition of the $\mathbf{F}, \mathbf{G}$-subformulae) and **(b)** a cyclic integer counter n. The transitions are determined by how the predictions and counters are allowed to change as described.

Next, we provide a formal definition for what it means to evaluate a formula with respect to a prediction:

Definition 4.1 (Evaluation). *For any formula $\varphi \in \text{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$ over P, a FG-prediction $\pi \in \Pi$ we inductively define a the propositional formula $[\,\varphi\,]_\pi$, the evaluation of a formula φ, as follows:*

$$[\,p\,]_\pi = p \qquad [\,\varphi \wedge \psi\,]_\pi = [\,\varphi\,]_\pi \wedge [\,\psi\,]_\pi \qquad [\,\mathbf{G}\psi\,]_\pi = \textit{true iff } \mathbf{G}\psi \in \alpha(\pi)$$
$$[\,\neg p\,]_\pi = \neg p \qquad [\,\varphi \vee \psi\,]_\pi = [\,\varphi\,]_\pi \vee [\,\psi\,]_\pi \qquad [\,\mathbf{F}\psi\,]_\pi = \textit{true iff } \mathbf{F}\psi \notin \alpha(\pi)$$

Next, to provide the reader with intution about the correctness of the construction we present two Propositions about the soundness and completeness of FG-predictions. First,

we observe that if π is sound in the sense that every formula $\mathbf{G}\psi \in \alpha(\pi)$ and $\mathbf{F}\psi \notin \alpha(\pi)$ is true at present then $[\,\varphi\,]_\pi$ evaluates to true with respect to w_0 indicates that φ is true for w.

Proposition 4.2 (Soundness). *For any formula $\varphi \in \mathrm{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$, a FG-prediction $\pi \in \Pi$ and word $w \in \Sigma^\omega$ if the following implications hold*

$$\mathbf{G}\psi \in \alpha(\pi) \implies w \vDash \mathbf{G}\psi \qquad \mathbf{F}\psi \notin \alpha(\pi) \implies w \vDash \mathbf{F}\psi$$

for every $\mathbf{G}\psi$, $\mathbf{F}\psi$ that is a subformula of φ then: $w_0 \vDash [\,\varphi\,]_\pi$ implies $w \vDash \varphi$.

Similarly, if π is complete in the sense that every $\mathbf{G}\psi$ that is true is in $\alpha(\pi)$ and every $\mathbf{F}\psi$ that is true is not in $\alpha(\pi)$ then φ is true for w implies $[\,\varphi\,]_\pi$ evaluates to true on w_0.

Proposition 4.3 (Completeness). *For any formula $\varphi \in \mathrm{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$, a FG-prediction $\pi \in \Pi$ and a word $w \in \Sigma^\omega$ if the following implications holds*

$$w \vDash \mathbf{G}\psi \implies \mathbf{G}\psi \in \alpha(\pi) \qquad w \vDash \mathbf{F}\psi \implies \mathbf{F}\psi \notin \alpha(\pi)$$

for every $\mathbf{G}\psi$, $\mathbf{F}\psi$ that is a subformula of φ then: $w \vDash \varphi$ implies $w_0 \vDash [\,\varphi\,]_\pi$.

Both the Propositions above can be proved by induction on the structure of φ. They are used in the correctness proof the construction. We use $\mathbb{F}(S)$ to denote all $\mathbf{F}\psi$ in set S. Finally, we give the formal definition of the construction for $\mathrm{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$ (devoid of $\mathbf{X}$s).

Definition 4.2 (Construction for $\mathrm{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$). *Given a formula φ in $\mathrm{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$ defined over propositions P, let $\mathcal{D}(\varphi)$ be the NBA (Q, I, δ, F) over the alphabet 2^P defined as follows*

- *Q is the set $\Pi \times [z]$, consisting of guess-counter pairs where $z = |\mathbb{F}_\varphi|+1$*
- *δ is the set of all transitions*

  $$(\pi_1, m) \xrightarrow{\sigma} (\pi_2, n)$$

  such that

  (A) *$\alpha(\pi_1) \subseteq \alpha(\pi_2)$ and $\gamma(\pi_1) = \gamma(\pi_2)$*

  (B) *for each $\psi \in \Psi(\pi_1, \pi_2)$, $\sigma \vDash [\,\psi\,]_{\pi_1}$*

(C) *n is updated as follows*

$$n = \begin{cases} m{+}1 \pmod{k}, & \sigma \vDash [\,\psi_m\,]_{\pi_1} \\ m & \text{otherwise} \end{cases}$$

where $k = |\mathbb{F}(\gamma)| + 1$, $\{\mathbf{F}\psi_1, \ldots, \mathbf{F}\psi_k\}$ is an enumeration of $\mathbb{F}(\gamma)$, $\psi_0 = \mathbf{tt}$.

Note: Since $\gamma(\pi_1) = \gamma(\pi_2)$ from (A) we ignore π_i when mentioning γ.

- *I is the set of transitions of the form $(\pi, 0) \xrightarrow{\sigma} (\pi', i)$ where $\sigma \vDash [\,\varphi\,]_\pi$ is true*
- *F is the set of states $(\pi, 0)$ where $\beta(\pi)$ is empty.*

Next, we present the theorem that states the correctness of the above construction.

Theorem 4.1. *For any formula $\varphi \in \mathrm{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$, the NBA $\mathcal{D}(\varphi)$ is a limit deterministic automaton of size $2^{\mathcal{O}(|\varphi|)}$ such that $L(\mathcal{D}(\varphi)) = [\![\varphi]\!]$.*

Proof. The number of states in $\mathcal{D}(\varphi)$ is bounded by $3^{|\mathbb{F}\cup\mathbb{G}|} \times |\mathbb{F}|$ and so clearly the size of $\mathcal{D}(\varphi)$ is exponential in $|\varphi|$.

We can see that $\mathcal{D}(\varphi)$ is limit deterministic as follows: The final states are of the form $(\pi, 0)$ where $\beta(\pi)$ is empty. Note that according to condition (A), $\beta(\pi)$ remains empty once it becomes empty, and $\alpha(\pi)$ and $\gamma(\pi)$ remain fixed. Hence the guess π can never change after visiting a final state. And since the counter is updated deterministically we have that any state reachable from a final state chooses its next state deterministically. We omit the proof of the fact $L(\mathcal{D}(\varphi)) = [\![\varphi]\!]$ since this result is going to be generalized in the construction for full LTL in Definition 4.7 for which we will provide a proof. □

Illustration

We illustrate the construction using once again the formula $\varphi = \mathbf{G}(a \vee \mathbf{F}b)$ for which the automaton is presented in Figure 4.1b. Note that every state has the formula φ present in α becuase any inital state has to evaluate φ to true and since it is a $\mathbf{G}$ formula it has to be in α and once assigned to be α it cannot be changed. Hence all states are initial. Next observe that we have two components owing to the two different γ: $\emptyset$ (in q_0, q_1) or $\{\mathbf{F}b\}$ (in q_2, q_3). In the states q_0, q_1, the subformula $\mathbf{F}b$ is in β, α respectively. We do not need

a counter for this component as γ is empty and hence shown to be always 0. There is a transtion from q_0 to q_0 where the FG-prediction hasn't changed, but we need to verify that $\psi = (a \vee \mathbf{F}b)$ is true (for $\varphi \in \alpha$) at the initial q_0 which is done by observing that $\mathbf{F}b$ is predicted to be β implying the truth of ψ. The state q_0 is non-final as β is non-empty. There is a transition from q_0 to q_1 which changes the prediction for $\mathbf{F}b$ and this forces only those transitions to be enabled where b is true (if b were replaced by a more complicated formula its truth would be enforced using a combination of the input being seen and the predictions made for the smaller temporal subformulae). q_1 has a transition to q_1 only enabled when a is true because at this point $\mathbf{F}b$ is predicted to be in α and hence assumed to be false. In q_2 and q_3 the predictions don't change only the counter does. In both states since $\mathbf{F}b \in \gamma$, we get $a \vee \mathbf{F}b$ to be evaluated to be true irrespective of the input being seen therefore $\varphi \in \alpha$ is automatically checked. The only remaining thing is $\mathbf{F}b \in \gamma$ which is done using a counter. When the counter is 0 it is forced to be incremented and when the counter is non-zero (in this case 1) it is incremented when b is evaluated to be true, once again if b were replaced by a more complicated formula its truth would have been derived using the next input and the prediction at that state. It is easy to see that this automaton is indeed limit deterministic and correctly accepts $[\![\varphi]\!]$.

Compositional Construction for LTL$\backslash GU$

In [29, 30] we also consider the fragment LTL$\backslash GU$ which is defined as follows:

Definition 4.3 (LTL\GU Syntax). *The fragment* LTL$\backslash GU$ *is given by the syntax*

$$\psi \quad ::= \quad \varphi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi \,\mathcal{U}\, \psi \qquad\qquad \varphi \in \mathrm{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$$

LTL$\backslash GU$ allows for Untils ($\mathcal{U}$) in addition to $\mathbf{F}, \mathbf{G}, \mathbf{X}$ as the temporal operator, but the untils are restricted to be outside the scope of any $\mathbf{G}$ subformula (Note that $\mathbf{F}$s outside the scope of $\mathbf{G}$ can be rewritten as a $\mathcal{U}$). We provide a compositional style construction for this fragment that produces exponential sized automata, where we compose a *master* and a *slave* automata to obtain the required one. The master automaton assumes that the truth values of the maximal until-free subformulae are known at each step and checks whether the

top-level until formula holds true. The master works over an extended set of propositions where the new propositions are introduced in place of the until-free subformulae. The slave automaton works over the original set of propositions and outputs at each step the truth value of the subformulae abstracted by the master in the form of the new propositions. The master and the slave are then composed such that they work together to check the entire formula. The master is produced using the standard NBA construction [59] and the slave is a *mealey* automaton which is a generalization of the construction presented in Definition 4.2. The interested reader should refer to [29] for details. We omit the details from this thesis because in the next section we provide a direct translation (as opposed to a compositional style) that works for full LTL while being efficient for a richer fragment LTL_{D}. We mention the construction for $\text{LTL}\backslash GU$ here because we will also be proving that the fragment LTL_{D} is semantically richer than $\text{LTL}\backslash GU$, thus justifying the worth of the direct construction.

4.2 Translation for full LTL

In this Section we present a generalization of the construction for $\text{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$ (Definition 4.2) that translates full LTL to limit deterministic Büchi automata. In Section 4.3 we shall see how this construction is not only optimal for $\text{LTL}\backslash GU$ but for a richer fragment LTL_{D}.

Definition 4.4 (LTL_{D} Syntax). *The formulae in the fragment* LTL_{D} *are given by the syntax for* ϑ*:*

$$\begin{aligned} \psi &::= \varphi \mid \psi \vee \varphi \mid \varphi \vee \psi \mid \psi \wedge \psi \mid \psi \,\mathcal{U}\, \varphi \mid \mathbf{G}\psi \mid \mathbf{X}\psi \qquad \varphi \in \text{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee) \\ \vartheta &::= \psi \mid \vartheta \vee \vartheta \mid \vartheta \wedge \vartheta \mid \vartheta \,\mathcal{U}\, \vartheta \mid \mathbf{X}\vartheta \end{aligned}$$

In the remainder of this chapter we will use the following terminology: *subformula of* φ is used to denote a node within the parse tree of φ. When we refer to the subformula as an LTL formula we will be referring to the formula at that node. Two subformulae that have the same formulae at their nodes need not be the same owing to the possibility of them being in different contexts. This distinction will be important as we treat formulae differently

depending on their contexts. For the purposes of describing different subfragments we qualify subformulae as being either *internal* or *external*.

Definition 4.5 (Scope). *A subformula ψ of φ is said to be* internal *if ψ is in the scope of some* $\mathbf{G}$*-subformula of φ, otherwise it is said to be* external.

LTL$(\mathbf{F}, \mathbf{G}, \wedge, \vee)$ allows for $\mathbf{G}$ and $\mathbf{F}$ as the only temporal operators. The fragment LTL$\backslash GU$ additionally allows for external $\mathcal{U}$ but not internal ones. Also, we choose to represent an external subformula of the form $\mathbf{F}\psi$ using until as $\mathbf{tt}\,\mathcal{U}\,\psi$. In other words every $\mathbf{F}$ will be internal. Unlike LTL$\backslash GU$, LTL_D allows for internal $\mathcal{U}$ but it is restricted. The following restrictions apply on LTL_D:

(i) The second argument of every internal $\mathcal{U}$ formula is in LTL$(\mathbf{F}, \mathbf{G}, \wedge, \vee)$,

(ii) At least one argument of every internal $\vee$ is in LTL$(\mathbf{F}, \mathbf{G}, \wedge, \vee)$

Note that LTL_D is strictly larger than LTL$\backslash GU$ in the syntactic sense, as every LTL$\backslash GU$ formula is also an LTL_D formula. We shall show in Section 4.4 that it is strictly richer in the semantic sense as well.

Next we define depth and height. A subformula ψ of φ is said to be at depth k if the number of $\mathbf{X}$ operators in φ within which ψ appears is exactly k. The height of a formula is the maximum depth of any of its subformulae.

4.2.1 Handling Untils and Nexts

First, we observe that the technique used for LTL$(\mathbf{F}, \mathbf{G}, \wedge, \vee)$ does not lend itself to the $\mathcal{U}\,/\mathbf{X}$ operators. The crucial property used above about $\mathbf{F}, \mathbf{G}$-formulae is that they cannot be simultaneously infinitely often true and infinitely often false unlike $\mathcal{U}\,/\mathbf{X}$ formulae. So if we tried the above technique for $\mathcal{U}\,/\mathbf{X}$ we would not get limit determinism since the truth of the $\mathcal{U}\,/\mathbf{X}$ formulae would have to be guessed infinitely often.

The key idea we use in handling $\mathcal{U}\,/\mathbf{X}$ is to propagate their *obligation* along the states. Let us say the automaton needs to check if a formula φ holds for an input w, and it begins by making an FG-prediction π about w. The obligation when no input has been seen is φ. When the first symbol w_0 is seen it needs to update the obligation to reflect what "remains

to be checked" for the rest of the input $w[1]$, in order for $w \vDash \varphi$ to hold, assuming π is correct for w. The automaton can keep updating the obligation as it sees each input symbol. The claim will be that the obligation is never falsified iff $w \vDash \varphi$, given that π is correct. This brings up some questions:

1. How are we exploiting opportunities for non-determinism?
2. How is the obligation computed at each step?
3. How is π checked to be correct in the presence of $\mathcal{U}$ /$\mathbf{X}$s?

Exploiting non-determinism.

Being able to exploit non-determinism helps in reducing the size of the automaton we construct. So the question is: how are we exploiting any opportunities for non-determinism (albeit for finite time)? The answer is to update the obligation non-deterministically. Checking the formula $\psi_1 \,\mathcal{U}\, \psi_2$ presents us with two alternatives: either ψ_2 is true now or $\psi_1 \wedge \mathbf{X}(\psi_1 \,\mathcal{U}\, \psi_2)$ is true now. Similarly $\psi_1 \vee \psi_2$ brings up two alternatives. We can pick between the obligations of these two choices non-deterministically. But we should somehow ensure that we are only allowed to use this non-determinism finitely often. This is where we treat internal and external (Definition 4.5) $\mathcal{U}$ /$\vee$ subformulae differently. The observation is that external $\mathcal{U}$ /$\vee$ need to be checked for only a finite amount of time. Hence the disjunctive choice presented by them can be dispatched non-deterministically each time without worrying about violating limit determinism. To illustrate this point we show the standard NBA for the formula $\varphi = a \,\mathcal{U}\, (\mathbf{G}b)$ in Figure 4.2 which turns out to be limit deterministic owing to the fact that the $\mathcal{U}$ is external. In Figure 4.1a we saw that the standard construction for $\varphi = \mathbf{G}(a \vee \mathbf{F}b)$ resulted in a NBA that was not limit-deterministic, and one of the reasons is that the $\mathbf{F}$, which is a special form of $\mathcal{U}$, is internal. An internal $\mathcal{U}$ /$\vee$ may need to be checked infinitely many times and hence the choice should not be resolved non-deterministically, but carried forward as a disjunction of the obligations of the choices. Passing the choice forward without resolving it comes at a cost of a bigger state space, this is akin to the subset construction where all the choices are being kept track of.

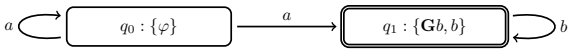


Figure 4.2: Standard NBA construction for $\varphi = a\,\mathcal{U}\,(\mathbf{G}b)$.

Now we begin to formalize the ideas. To exploit the non-determinism allowed by the external $\mathcal{U}\,/\vee$ we introduce the concept of *ex-choice*. We use Λ_φ to denote the set of all external $\mathcal{U}\,/\vee$ subformulae. Any subset of it $\lambda \subseteq \Lambda_\varphi$ is called an ex-choice. An ex-choice dictates how each external $\mathcal{U}\,/\vee$ should be satisfied if it needs to be satisfied. The interpretation associated with λ is the following: if $\psi_1\,\mathcal{U}\,\psi_2 \in \lambda$ then ψ_2 has to hold or if $\psi_1\,\mathcal{U}\,\psi_2 \in \Lambda_\varphi - \lambda$ then $\psi_1 \wedge \mathbf{X}(\psi_1\,\mathcal{U}\,\psi_2)$ has to hold. Similarly if $\psi_1 \vee \psi_2 \in \lambda$ then ψ_1 has to hold and if $\psi_1 \vee \psi_2 \in \Lambda_\varphi - \lambda$ then ψ_2 has to hold. The automaton we are going to construct is going to non-deterministically pick an ex-choice at each step and use it resolve the choices on external $\mathcal{U}\,/\vee$. After a finite time the ex-choice will not matter because the obligations will not consist of any external $\mathcal{U}\,/\vee$ that need to be checked (which will be enforced as a part of the acceptance condition), and hence limit determinism is ensured. The ex-choice picked along a transition is going to determine the obligation computed. Which leads us to the question of how the obligation is computed.

Computing Obligation.

We define the *derivative* of a formula μ w.r.t an input symbol σ, FG-prediction π and ex-choice λ. The derivative should capture the obligation/requirement on any word ρ such that those obligations are able to imply that

1. $\sigma\rho$ satisfies μ
2. $\sigma\rho$ respects the ex-choice λ.

This enables us to keep passing on the obligation forward as we see each symbol of the input by taking the derivative of the obligation so far. This requires us to ensure that the ex-choice λ picked when we are taking the transition dictates how a formula in Λ_φ should be satisfied

if it needs to be. With that in mind we define $f(\lambda)$ as follows:

$$f(\lambda) = (\wedge_{(\phi \mathcal{U} \psi \in \lambda)} \phi \mathcal{U} \psi \Rightarrow \psi) \wedge (\wedge_{(\phi \mathcal{U} \psi \in (\Lambda_\varphi - \lambda))} \phi \mathcal{U} \psi \Rightarrow (\phi \wedge \mathbf{X}(\phi \mathcal{U} \psi))) \\ \wedge (\wedge_{(\phi \vee \psi \in \lambda)} \phi \vee \psi \Rightarrow \phi) \wedge (\wedge_{(\phi \vee \psi \in \Lambda_\varphi - \lambda)} \phi \vee \psi \Rightarrow \psi) \tag{4.2}$$

Therefore the two requirements on $\sigma\rho$ above, simplifies to $\sigma \vDash \mu \wedge f(\lambda)$. But predictions made by π already tell us the truth of some of the subformulae, they need to be taken into account. This can be done using the evaluation function (Definition 4.1). Now we are ready to give a declarative definition of the derivative:

Definition 4.6. *Given an LTL formula μ over P, and a triple $\varepsilon = (\sigma, \pi, \lambda)$ where $\sigma \in 2^P$, $\pi \in \Pi(\varphi)$ and $\lambda \subseteq \Lambda_\varphi$: an LTL formula ψ is said to be a **derivative** of μ w.r.t to ε if*

$$\forall\, \rho \in \left(2^P\right)^\omega \quad \rho \vDash \psi \implies \sigma\rho \vDash [\, \mu \wedge f(\lambda)\,]_\pi$$

*The **weakest derivative** of μ w.r.t ε, denoted by $\nabla(\mu, \varepsilon)$, is a derivative such that $\psi \implies \nabla(\mu, \varepsilon)$ for any other derivative ψ.*

Since we will only be interested in the weakest derivative (as opposed to any other derivative) we shall refer to it as the derivative. The above definition is only declarative in the sense that it does not give us an explicit way to compute the derivative. We present this definition here for the sake of simplicity and ease of understanding for the reader. In Appendix A we provide a syntactic definition and all the necessary machinery that allows us to compute such a formula. The syntactic definition also restricts the representation of the obligations to $\mathcal{B}^+(\varphi)$ which is the set of all positive Boolean combinations of subformulae of φ.

The automaton now will have an extra component μ corresponding to the obligation along with (π, n) from before. In the initial state μ will be the given formula φ that needs to be checked. At each step, the automaton sees an input symbol σ and makes a non-deterministic ex-choice $\lambda \subseteq \Lambda_\varphi$. The obligation at the next state will then become $\nabla(\mu, \varepsilon)$ where $\varepsilon = (\sigma, \pi, \lambda)$. The process continues as long as the obligation is never falsified. In order to ensure that every external until is dispatched in finite time, we impose that the obligation μ in the final states is *ex-free*, i.e. free of any formulae in Λ_φ. When the obligation is ex-free

the ex-choice does not play a role in determining its derivative and we shall drop λ whenever that is the case, and this eliminates any non-determinism once a final state is visited. In order to ensure that an internal until, say $\phi \mathcal{U} \psi$ is not delayed forever, we involve $\mathbf{F}\psi$ in the FG-prediction and enhance the definition of substitution to say that $\phi \mathcal{U} \psi$ is replaced with $\mathbf{ff}$ if $\mathbf{F}\psi \in \alpha$. This way the derivative will impose that $\mathbf{F}\psi$ is true whenever $\phi \mathcal{U} \psi$ is claimed to be true. With this in mind we define the closure of φ, denoted by $\mathcal{C}(\varphi)$, to be set of all $\mathbf{F}, \mathbf{G}$-subformulae of φ, along with all $\mathbf{F}\psi$ for every internal $\phi \mathcal{U} \psi$ subformula of φ. We re-define an FG-prediction π to be any tri-partition of $\mathcal{C}(\varphi)$. Note that for every $\mathbf{F}\psi$ or $\mathbf{G}\psi$ in $\mathcal{C}(\varphi)$, ψ is internal.

Example 4.2. *Let* $\varphi = \mathbf{G}(\mathbf{F}a \vee (b \mathcal{U} c))$. *Here* $\mathcal{C}(\varphi) = \{\varphi, \mathbf{F}a, \mathbf{F}c\}$

Example 4.3. *Let* $\varphi = a \mathcal{U} (b \wedge \mathbf{G}c)$ *be an internal subformula of some given formula.* $\nabla(\varphi, \varepsilon)$ *can take different values depending upon* $\varepsilon = (\sigma, \pi)$. *Here ex-choice* λ *does not play a role because the only* $\mathcal{U}$ *is internal. Note that* $\varphi' = \mathbf{F}(b \wedge \mathbf{G}c)$ *is in the closure. If* $\varphi' \in \alpha$, *then* $\nabla(\varphi, \varepsilon) = \mathbf{ff}$ *because* $[\varphi]_\pi$ *would be* $\mathbf{ff}$ *owing to* φ *being substituted with* $\mathbf{ff}$. *Let* $\varphi' \notin \alpha$. *Now if* $\mathbf{G}c \in \alpha$ *then substituting* $\mathbf{tt}$ *in place of* $\mathbf{G}c$ *gives us* $a \mathcal{U} b$ *whose satisfaction depends upon the truth of* a *and* b *as given by* σ. *So if* $\sigma(b) = \mathbf{tt}$ *then the* $\mathcal{U}$ *is immediately satisfied and so* $\nabla(\varphi, \varepsilon) = \mathbf{tt}$. *If* $\sigma(b) = \mathbf{ff}$ *then the* $\mathcal{U}$ *is delayed and hence* $\nabla(\varphi, \varepsilon)$ *is either* $a \mathcal{U} b$ *or* $\mathbf{ff}$ *depending on* $\sigma(a) = \mathbf{tt}/\mathbf{ff}$ *respectively. If* $\mathbf{G}c \notin \alpha$ *then truth of* b *does not matter (as replacing* $\mathbf{G}c$ *with* $\mathbf{ff}$ *makes* $b \wedge \mathbf{G}c = \mathbf{ff}$*) and once again the derivative is* $\varphi/\mathbf{ff}$ *depending upon* $\sigma(a)$.

Checking FG-predictions in the presence of untils and nexts.

The main idea in being able to check an FG-prediction π was that a correct prediction about an $\mathbf{F}, \mathbf{G}$-subformula also tells us its truth. When we have $\mathcal{U}/\mathbf{X}$s in the mix, we no longer have a prediction available for them, and hence no immediate way to check if some subformula is true. For example when $\mathbf{G}\psi \in \alpha$ we needed to check ψ is true and we did so inductively using the predictions for subformulae in ψ. Now, since ψ can have $\mathcal{U}/\mathbf{X}$ within them it is not clear how we are going to check truth of ψ. In this case we pass ψ to the obligation μ. Similarly when the prediction of $\mathbf{F}\psi$ is changed from β to α we need to

check ψ is true so once again we pass ψ to the obligation. So we use the set of formulae Ψ defined in Equation 4.1, and update the obligation along a transition $(\mu, \pi, n) \xrightarrow{\sigma} (\mu', \pi', n')$ as: $\mu' = \nabla(\mu \wedge (\wedge_{\psi \in \Psi} \psi), \varepsilon)$ where $\varepsilon = (\sigma, \pi, \lambda)$. Now consider the case when the counter is $n > 0$ and need to verify that the n^{th} $\mathbf{F}\psi$ formula in γ is true. In this case we cannot pass on ψ to the obligation because $\mathbf{F}\psi$ may be true because ψ is true at a later point and not now. Since we cannot predict when ψ is going to be true, we carry the disjunction of all the derivatives of ψ since the counter was incremented to n. We keep doing it until this "carry" becomes true indicating that ψ became true at some point since we started checking for it. We also increment the counter at that point. This "carry" becomes yet another component ν in the automaton's state. We use $\mathbb{F}(S)$ to denote all $\mathbf{F}\psi$ in set S. Now we are ready to put the pieces together to formally describe the entire construction.

Definition 4.7 (Construction). *Given a formula $\varphi \in$ LTL over propositions P, let $\mathcal{D}(\varphi)$ be the NBA (Q, δ, I, F) over the alphabet 2^P defined as follows:*

- Q *is the set* $\mathcal{B}^+(\varphi) \times \mathcal{B}^+(\varphi) \times \Pi(\varphi) \times [n]$ *where* $n = |\mathbb{F}(\mathcal{C}(\varphi))| + 1$
- δ *is the set of all transitions* $(\mu, \nu, \pi, m) \xrightarrow{\sigma} (\mu', \nu', \pi', m')$ *such that*

  (a) $\alpha(\pi) \subseteq \alpha(\pi')$ *and* $\gamma(\pi) = \gamma(\pi')$

  (b) $\mu' = \nabla(\mu \wedge \theta, \varepsilon)$ *for some* $\lambda \subseteq \Lambda_\varphi$

  where $\theta = (\wedge_{\psi \in \Psi} \psi)$, Ψ *as defined in (4.1) and* $\varepsilon = (\sigma, \pi, \lambda)$

  (c) $m' = \begin{cases} (m+1) \pmod{|\mathbb{F}(\gamma)| + 1} & \nu = \mathbf{tt} \\ m & \text{otherwise} \end{cases}$

  (d) $\nu' = \begin{cases} \psi_{m'} & \nu = \mathbf{tt} \\ \nabla(\nu, \varepsilon) \vee \psi_m & \text{otherwise} \end{cases}$

  where $\{\mathbf{F}\psi_1, .., \mathbf{F}\psi_k\}$ *is an enumeration of* $\mathbb{F}(\gamma)$, $\psi_0 = \mathbf{tt}$ *and* $\varepsilon = (\sigma, \pi)$

- I *is all states of the form* $(\varphi, \mathbf{tt}, \pi, 0)$

■ *F is all states of the form $(\mu, \mathbf{tt}, \pi, 0)$ where $\beta(\pi) = \emptyset$, $\mu \neq \mathbf{ff}$, μ is ex-free*

We state the correctness result here, the proofs can be found in Appendix A.

Theorem 4.2. *For $\varphi \in$ LTL, $\mathcal{D}(\varphi)$ is a limit deterministic automaton such that $L(\mathcal{D}(\varphi)) = [\![\varphi]\!]$ and $\mathcal{D}(\varphi)$ is of size at most double exponential in φ.*

The number of different formulae in $\mathcal{B}^+(\varphi)$, is at most double exponential in the size of φ, since each can be represented as a collection of subsets of subformulae of φ. $\Pi(\varphi)$ is simply tripartition of $\mathcal{C}(\varphi)$ which is bounded above by $3^{|\varphi|}$. And the counter can take $|\mathbb{F}(\mathcal{C}(\varphi))| + 1$ different values which is $\leq |\varphi|$. The entire state space $\mathcal{B}^+(\varphi) \times \mathcal{B}^+(\varphi) \times \Pi(\varphi) \times [n]$ is upper bounded by the product of these which is clearly doubly exponential.

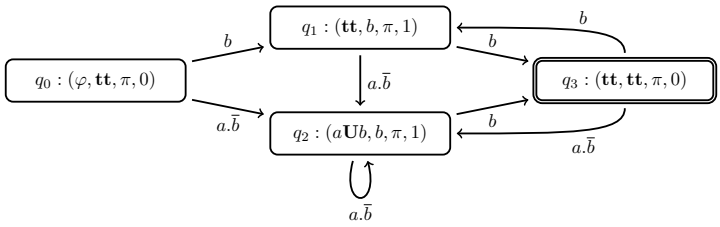


Figure 4.3: Our construction for $\varphi = \mathbf{G}(a \,\mathcal{U}\, b)$.

Illustration

We illustrate our construction using $\varphi = \mathbf{G}(a \,\mathcal{U}\, b)$ which is a formula outside LTL$\backslash GU$. The automaton for φ is shown in Figure 4.3. First note that the $\mathcal{C}(\varphi) = \{\varphi, \mathbf{F}b\}$. Next, observe that the only interesting FG-prediction is π in which $\alpha = \{\varphi\}$, $\beta = \emptyset$ and $\gamma = \{\mathbf{F}b\}$. This is because any initial state will have $\mu = \varphi$ which forces $\varphi \in \alpha$, and since predictions in α don't change, every reachable state will have $\varphi \in \alpha$ as well. As for $\mathbf{F}b$ note that the corresponding internal until $a \,\mathcal{U}\, b$ will become $\mathbf{ff}$ if $\mathbf{F}b$ is in α and thus making the derivative $\mathbf{ff}$ ($a \,\mathcal{U}\, b$ is added to the obligation at each step since $\varphi \in \alpha$ and rule **(b)**). Therefore $\mathbf{F}b$ cannot be in α, and it cannot be in β because then it would be eventually in α. So $\mathbf{F}b$ has to be in γ. Now that π is fixed, and given input σ, the obligation μ changes according to rule **(b)** as $\mu' = \nabla(\mu \wedge (a \,\mathcal{U}\, b), (\sigma, \pi))$. Similarly the carry ν changes to b if $\nu = \mathbf{tt}$ (as in q_3 to q_1/q_2) and becomes $\nu' = \nabla(\nu, (\sigma, \pi)) \vee b$ otherwise in accordance with rule **(d)**. The

initial state is q_0 with $\mu = \varphi$, $\nu = \mathbf{tt}$ and counter $= 0$. The counter is incremented whenever ν becomes $\mathbf{tt}$. It is easy to see that the automaton indeed accepts $\mathbf{G}(a\,\mathcal{U}\,b)$ and is limit deterministic.

4.3 Optimality for $\mathrm{LTL_D}$

In this section we state the results regarding the efficiency of our construction for $\mathrm{LTL_D}$. We prove that there are only exponentially many reachable states in $\mathcal{D}(\varphi)$. A state $q = (\mu, \nu, \pi, n)$ of $\mathcal{D}(\varphi)$ is called reachable if there exists a valid finite run of the automaton that ends in q. A μ is said to be reachable if (μ, ν, π, n) is reachable for some choice of ν, π and n. Similarly for ν. We show that the space of reachable μ and ν is only exponentially large in the size of φ. Our approach will be to show that every reachable μ (or ν) can be expressed in a certain way, and we will count the number of different such expressions to obtain an upper bound. The expression for μ and ν relies on them being represented in DNF form and uses the syntactic definition of the derivative given in Section A.2 of the Appendix. Therefore we state only the main result here and present the proofs in Section A.4 of the Appendix.

Theorem 4.3. *For $\varphi \in \mathrm{LTL_D}$ the number of reachable states in the $\mathcal{D}(\varphi)$ is at most exponential in $|\varphi|$.*

4.4 Expressive Power of $\mathrm{LTL_D}$

In this section we show that $\mathrm{LTL_D}$ is semantically more expressive than $\mathrm{LTL}\backslash GU$. We demonstrate that the formula $\varphi_0 = \mathbf{G}(p \vee (q\,\mathcal{U}\,r))$ which is expressible in $\mathrm{LTL_D}$, cannot be expressed by any formula in $\mathrm{LTL}\backslash GU$.

Let us fix integers $\ell, k \in \mathbb{N}$. We will use $\mathrm{LTL}_\ell(F, G)$ to denote the subfragment of $\mathrm{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$ where formulae have maximum height ℓ. Since $\mathbf{X}$ distributes over all other operators we assume that all the $\mathbf{X}$s are pushed inside. We use $\mathrm{LTL}_{\ell,k}\backslash GU$ to denote the fragment where formulae are built out of $\mathcal{U}$, $\wedge$, $\vee$ and $\mathrm{LTL}_\ell(F, G)$ formulae such that the number of $\mathcal{U}$s used is at most k.

Next, consider the following strings over 2^P where $P = \{p, q, r\}$:

$$u = \{p\}\{p,q\}^{\ell}\{p\} \quad v = \{q\}\{p,q\}^{\ell}\{r\} \quad w = \{q\}\{p,q\}^{\ell}\{p\}$$
$$s_k = (uv)^{k+1}u \qquad \sigma = (uv)^{\omega} \qquad \eta_k = s_k wv\sigma \tag{4.3}$$

The observation we make is that σ satisfies φ_0 but η_k does not, stated in Theorem 4.4. In order to prove our main result we will need a proposition and a lemma. We will use $\text{suf}(w)$ to denote the set of all suffixes of the word w.

Proposition 4.4. *For any $\rho \in \text{suf}(\eta_k)$ one of the following holds*

1. *Either $\rho \in \text{suf}(\sigma)$, or*

2. *$\exists\, x \in \text{suf}(s_k)$ such that $\rho = xwv\sigma$*

For $\rho \in \text{suf}(\eta_k)\backslash\text{suf}(\sigma)$, i.e., ρ is of the form $xwv\sigma$ where $x \in \text{suf}(s)$, let $cut_w(xwv\sigma) = xv\sigma \in \text{suf}(\sigma)$. What we are trying to say is that for all suffixes of η_k that are not suffixes of σ, cut_w removes the substring w. Next we show that every suffix of η_k is logically equivalent (w.r.t $\text{LTL}_\ell(F, G)$) to some suffix of σ.

Lemma 4.1. *For every $\psi \in \text{LTL}_\ell(F, G)$, for any k and any $\rho \in \text{suf}(\eta_k)\backslash\text{suf}(\sigma)$:*

$\rho \vDash \psi$ *iff* $cut_w(\rho) \vDash \psi$

Proof. ρ is of the form $xwv\sigma$. We perform induction on $|x|$ to prove the required statement.

Base Case: $x = \epsilon$, i.e $\rho = wv\sigma$. We prove the base case by induction on ψ. Observe that for every ψ of the form $\mathbf{X}^i a$ where $a \in \{p, q, r\}$ and $0 \leq i \leq \ell$, we have $wv\sigma \vDash \psi$ iff $v\sigma \vDash \psi$. For the inductive step the only intersting cases are when ψ is $\mathbf{F}$ or $\mathbf{G}$ formula. Consider $\psi = \mathbf{F}\psi_1$

$$\begin{aligned}
v\sigma \vDash \mathbf{F}\psi_1 &\implies \exists y \in \text{suf}(v\sigma) \text{ s.t } y \vDash \psi_1 \\
&\implies wv\sigma \vDash \mathbf{F}\psi_1 \text{ becaue } y \in \text{suf}(wv\sigma) \\
wv\sigma \vDash \mathbf{F}\psi_1 &\implies \text{either (a) } wv\sigma \vDash \psi_1 \Rightarrow v\sigma \vDash \psi_1 \text{ (ind hyp)} \Rightarrow v\sigma \vDash \mathbf{F}\psi_1 \\
&\qquad \text{or (b) } \exists y \in \text{suf}(wv\sigma)\backslash\{wv\sigma\} \text{ s.t } y \vDash \psi_1 \\
&\qquad\quad \Rightarrow v\sigma \vDash \mathbf{F}\psi_1 \text{ because } y \in \text{suf}(v\sigma)
\end{aligned}$$

We continue the induction on ψ by considering the case when $\psi = \mathbf{G}\psi_1$

$$\begin{aligned}
v\sigma \vDash \mathbf{G}\psi_1 &\implies \forall y \in \text{suf}(v\sigma), y \vDash \psi_1 \\
&\implies \forall y \in \text{suf}(wv\sigma)\backslash\{wv\sigma\}, y \vDash \psi_1 \text{ and} \\
&\qquad wv\sigma \vDash \psi_1 (\text{since } v\sigma \vDash \psi_1 \text{ and ind hyp}) \\
&\implies wv\sigma \vDash \mathbf{G}\psi_1 \\
wv\sigma \vDash \mathbf{G}\psi_1 &\implies \forall y \in \text{suf}(wv\sigma), y \vDash \psi_1 \\
&\implies \forall y \in \text{suf}(v\sigma), y \vDash \psi_1 (\text{suf}(v\sigma) \subseteq \text{suf}(wv\sigma)) \\
&\implies v\sigma \vDash \mathbf{G}\psi_1
\end{aligned}$$

Returning back to the induction on x consider $\rho = xwv\sigma$ where $x = ay$ with $a = \{p\}$, $\{q\}$, $\{r\}$ or $\{p, q\}$. The proof is again by structural induction on ψ and similar to the base case hence we omit it. □

In order to prove Theorem 4.4 we prove the following stronger statement: For $\varphi \in \text{LTL}_{\ell,k}\backslash GU$, $x \in \text{suf}(uv)$, $j \geq k$: if $x\sigma \vDash \varphi$ then $x(uv)^j uwv\sigma \vDash \varphi$. We perform induction on k. The base case is when $k = 0$ i.e φ has no $\mathcal{U}$, and it directly follows from the previous Lemma. For the inductive case consider $k = n + 1$. The interesting case is when φ is of the form $\psi_1 \mathcal{U} \psi_2$. The first position i along $x\sigma$ where ψ_2 holds has to be $< |uv|$. Consider the suffix of $x(uv)^m uwv\sigma$ at position i , using induction hypothesis we can conclude that ψ_2 holds at that position. Similarly for every prefix of $x(uv)^m uwv\sigma$ that begins before position i we can conclude that it satisfies ψ_1 using induction hypothesis.

Theorem 4.4. $\forall\, \varphi \in \text{LTL}_{\ell,k}\backslash GU \quad \sigma \vDash \varphi \implies \eta_k \vDash \varphi$

Corollary 4.1. *φ_0 is not expressible in $\text{LTL}_{\ell,k}\backslash GU$. Also since ℓ and k are arbitrary, φ_0 is not expressible in $\text{LTL}\backslash GU$.*

4.5 Model Checking MDPs

In this section we comment on the impact of our constructions on the qualitative model checking problem. We observe that our exponential sized construction for LTL_D yields a

EXPTIME algorithm for qualitative model checking problem for MDPs, as compared to the previously known 2EXPTIME upper bound. We also prove an EXPTIME lower bound for the fragment LTL$\backslash GU$ thus showing that the problem is in fact EXPTIME-complete.

4.5.1 Model Checking complexity for $\mathrm{LTL_D}$

Proposition 4.5. *The qualitative model checking problem for MDPs against specification in* $\mathrm{LTL_D}$ *can be solved in time exponential in the specification and linear in the size of the MDP*

Proof. Follows from our construction being of exponential size and the fact that the model checking of MDPs can be done by performing a linear time analysis of the synchronous product of the MDP and the automaton (Proposition 2.1). □

Proposition 4.6. *The qualitative model checking problem for MDPs against specification in* $\mathrm{LTL}(\mathbf{F}, \mathbf{X}, \wedge)$ *is EXPTIME-hard, as a conseuqnce it is EXPTIME-hard for every fragment that contains* $\mathrm{LTL}(\mathbf{F}, \mathbf{X}, \wedge)$ *(which includes* $\mathrm{LTL_D}$*).*

Proof. We reduce the membership problem for polynomial space-bounded alternating Turing Machines to the the problem of qualitative model checking of MDPs against $\mathrm{LTL}(\mathbf{F}, \mathbf{X}, \wedge)$. Recall that an alternating Turing Machine T over tape alphabet Σ is a specified as a tuple (Q, δ, q_0, g) where Q is the set of states, $\delta : Q \times \Sigma \to \mathcal{P}(Q \times \Sigma \times \{-1, +1\})$ is the transition function, q_0 is the initial state and $g : Q \to \{\exists, \forall, \mathsf{acc}, \mathsf{rej}\}$ categorizes each state into one of 4 types, namely: existential, universal, accept and reject. A configuration of the T is a triple $(q, w, i) \in Q \times \Sigma^* \times \mathbb{N}$ where q is the current state, w is the content of the tape and $i \in \{0, 1, \ldots, |w| - 1\}$ is the head position. In a $s(n)$ space -bounded ATM the tape content w is always restricted to be of size $s(|x|)$ where x is the input word. The initial configuration on input x is $(q_0, x, 0)$. The computation of the ATM is a sequence of configurations which can be viewed as a two player game between $\exists$-player and $\forall$-player. When the configuration is in an existential state ($g(q) = \exists$) it is $\exists$-player's turn to pick a transition and when it is in a univeral state ($g(q) = \forall$) it is $\forall$-player's turn to pick the next transition. The transition modifies the configuration in the usual way a Turing Machine operates (change state, write on the head position and move the tape head left/right) according to δ. The play is continued until a state is reached which is accepting or rejecting (together known as the halting states).

If the play ends in an accept state then $\exists$-player wins and if it ends in a reject state then $\forall$-player wins. The input word x is said to be accepted iff $\exists$-player has a winning strategy from the initial configuration.

In our reduction we consider polynomial space-bounded ATMs. Without loss of generality we can assume that the ATM halts on every input because polynomial space allows for counting the number of steps, and so an $s(n)$ space-bounded machine can force the play to halt after $2^{s(n)}$ steps. Given polynomial space-bounded ATM T and an input x we construct an MDP $\mathcal{M}$ and formula $\varphi \in \text{LTL}(\mathbf{F}, \mathbf{X}, \wedge)$ such that T accepts x iff $\exists \mathfrak{S} : \Pr^{\mathfrak{S}}_{\mathcal{M}}(\varphi) > 0$.

Reduction sketch: The MDP $\mathcal{M}$ is going to be such that the scheduler plays the role of the $\exists$-player and the stochastic choices play the role of the $\forall$-player. Loosely speaking, the states of $\mathcal{M}$ encode information about a configuration and the transitions of $\mathcal{M}$ encode how configurations change. The scheduler then attempts to reach a state corresponding to a configuration that is accepting. If $\exists$-player has a winning strategy for input x then there is a scheduler that reaches an accepting configuration with probability 1, because no matter what stochastic choices are made (which correspond to strategies of $\forall$-player), $\exists$-player can force the play into an accepting state. But on the other hand if $\exists$-player does not have a winning strategy for input x then no scheduler can win with probability 1, but some scheduler might still be able to reach accepting state with probability > 0 because the stochastic player cannot make intentional choices like the $\forall$-player. This is a problem because if T rejects input x we want no scheduler to be able to reach an accept state with probability > 0. To rectify this problem the MDP $\mathcal{M}$ is designed to repeatedly orchestrate the game until a reject state is reached. In this scenario if $\exists$-player has a winning strategy then there is a scheduler corresponding to the strategy which will reach the accepting state on each round of the game with probability 1, and if $\forall$-player has a winning strategy then every scheduler has non-zero probability of reaching the reject state in one round and hence has 0 probability of avoiding the reject state when played forever.

The issue we have not discussed so far is the size of the MDP $\mathcal{M}$. If we are to have a state for each different configuration then $\mathcal{M}$ would have to be exponentially large owing to the tape being polynomially long and the reduction would be incorrect. Therefore, instead of storing the entire configuration, we only store the important details about the configuration.

The tape contents of the configuration are generated sequentially, and the responsibility for doing so is handed to the scheduler. But the scheduler has a conflicting interest in reaching the accept state and might cheat, i.e., make an incorrect guess for its advantage. The specification φ is then strengthened to prevent the scheduler from cheating.

Formal details: In the MDP $\mathcal{M}$ that we construct, each state is going to be either non-deterministic or stochastic: a state is called non-deterministic if every enabled action on that state has exactly one successor state, and a state is called stochastic if it has no more than one enabled action. The names of the actions are irrelevant and will be omitted in the description. We view each state of $\mathcal{M}$ as a tuple $(\mathbf{q}, (\mathbf{p}, \sigma_{\mathbf{p}}), (\mathbf{h}, \sigma_{\mathbf{h}}), (\mathbf{w}, \sigma_{\mathbf{w}}))$ where $\mathbf{q}$ is the state of the current configuration, $\mathbf{p}$ is the index of the cell being guessed, $\sigma_{\mathbf{p}}$ is the symbol at position $\mathbf{p}$ in the current configuration, $\mathbf{h}$ is the index of the head in the current configuration, $\sigma_{\mathbf{h}}$ is the symbol at position $\mathbf{h}$ (underneath the head) in the current configuration, $\mathbf{w}$ is the index of the cell at which the tape was in the previous configuration (the cell to be written), and $\sigma_{\mathbf{w}}$ is the symbol at position $\mathbf{w}$ in the current configuration. We divide the state space of $\mathcal{M}$ into two: initial part $\mathcal{I}$ and the non-initial part $\mathcal{S}$. $\mathcal{I}$ is used to spell out the starting configuration, and the $\mathcal{S}$ is used to guess the configurations that follow.

First, we describe how $\mathcal{I}$ and $\mathcal{S}$ generate configurations. The initial part $\mathcal{I}$ consists of s_x (short hand for $s(|x|)$, where s is the polynomial bound on the space) different states of the form $(q_0, (i, x_i), (0, x_0), (\bot, \bot))$ for $i \in [0, s_x)$. Here $\mathbf{q}$ is fixed to be the inital state q_0, $(\mathbf{h}, \sigma_{\mathbf{h}})$ is fixed to be $(0, x_0)$ indicating that the tape head is over the first cell of the tape, and $(\mathbf{w}, \sigma_{\mathbf{w}})$ is fixed to be $(\bot, \bot)$ indicating that there is no previous configuration. The pair $(\mathbf{p}, \sigma_{\mathbf{p}})$ traverses over the initial input tape (x buffered by blank symbols to make it of length s_x). The transitions in $\mathcal{I}$ are of the form $(q_0, (i, x_i), (0, x_0), (\bot, \bot)) \to (q_0, (i{+}1, x_{i+1}), (0, x_0), (\bot, \bot))$ for $i \in [0, s_x{-}1)$, indicating that the cells of the tape are being traversed from left to right. All the states in $\mathcal{I}$ except possibly the last ones (where $\mathbf{p} = s_x{-}1$) are deterministic (only one enabled action with a single successor). In the non-initial part $\mathcal{S}$ the states consists of all possible assignments to the symbolic tuple $(\mathbf{q}, (\mathbf{p}, \sigma_{\mathbf{p}}), (\mathbf{h}, \sigma_{\mathbf{h}}), (\mathbf{w}, \sigma_{\mathbf{w}}))$. All the states in $\mathcal{S}$ except possibly the last ones (where $\mathbf{p} = s_x{-}1$) are non-deterministic. The only transitions allowed within this part are of the form $(q, (i, \sigma), (h, \sigma_h), (w, \sigma_w)) \to (q, (i{+}1, \sigma'), (h, \sigma_h), (w, \sigma_w))$.

Here the components $\mathbf{q}, (\mathbf{h}, \sigma_{\mathbf{h}})$ and $(\mathbf{w}, \sigma_{\mathbf{w}})$ remain fixed across transitions, and $(\mathbf{p}, \sigma_{\mathbf{p}})$ is allowed to change indicating that the tape is being traversed left to right.

Next we describe transitions of $\mathcal{M}$ once a configuration has been generated (in $\mathcal{I}$ or $\mathcal{S}$). When $\mathbf{p} = s_x{-}1$ it is time to guess details of the next configuration. So consider the state to be of the form $(q, (s_x{-}1, \sigma), (h, \sigma_h), (w, \sigma_w))$. The next configuration depends on whose turn it is to play in the game.

- If q is an existential (universal) state then the next value for $\mathbf{q}$ is chosen non-determinstically (stochastically) from the available transitions in δ. The choice of this transition also dictates the next value of $\sigma_{\mathbf{w}}$, and the direction in which to move the tape head. Thus transitions here are of the form $(q, (s_x{-}1, \sigma), (h, \sigma_h), (w, \sigma_w)) \to (q', (0, \sigma'), (h{+}\iota, \sigma_{h'}), (h, \sigma_{w'}))$, where $(q', \sigma_{w'}, \iota) \in \delta(q, \sigma_h)$. The symbols σ', $\sigma_{h'}$ are guessed non-deterministically by the scheduler.

- If q is an accept state then this round has ended in the scheduler's favour and the next round is started from the first state in $\mathcal{I}$.

- If q is a reject state then no more rounds are played and that state of the MDP is turned into an absorbing state.

Next, we look at the design of specification φ, and see how it enforces consecutive conifgurations to be valid. First, observe that for a non-halting configuration the contents of a cell located at a position other than the head remains unchanged as it moves to the next configuration. We enforce this by using the following formula:

$$\varphi_1 = \bigwedge_{a \in \Sigma} \mathbf{G}\left((\mathbf{p} \neq \mathbf{h} \wedge g(\mathbf{q}) \in \{\exists, \forall\}) \Rightarrow ((\sigma_{\mathbf{p}} = a) \Rightarrow \mathbf{X}^{s_x}(\sigma_{\mathbf{p}} = a))\right) \tag{4.4}$$

Note that the $\mathbf{p} \neq \mathbf{h}$, $g(\mathbf{q}) \in \{\exists, \forall\}$ and $\sigma_{\mathbf{p}} = a$ are all propositions whose truth can be derived by looking at the state alone. Next, we need to ensure that the symbol $\sigma_{\mathbf{w}}$ guessed for cell $\mathbf{w}$ matches with the symbol that is generated when $\mathbf{p}$ takes up the value of $\mathbf{w}$ in that configuration, and this is done using formula $\varphi_2 = \mathbf{G}\left((\mathbf{p} = \mathbf{w}) \Rightarrow (\sigma_{\mathbf{p}} = \sigma_{\mathbf{w}})\right)$. Similarly, we need to ensure that the symbol $\sigma_{\mathbf{h}}$ guessed for cell $\mathbf{h}$ in a configuration matches with actual symbol generated at position corresponding to $\mathbf{h}$, this is achieved using $\varphi_3 =$

$\mathbf{G}\left((\mathbf{p} = \mathbf{h}) \Rightarrow (\sigma_{\mathbf{p}} = \sigma_{\mathbf{h}})\right)$. Finally, we have formula $\varphi_4 = \mathbf{G}(g(\mathbf{q}) \neq \mathsf{rej})$ to specify that we are looking for a scheduler that can win every round of play. Putting everything together we have $\varphi = \wedge_{i=1}^{4} \varphi_i$ as the final specification which is in LTL$(\mathbf{F}, \mathbf{X}, \wedge)$. □

Theorem 4.5 (Complexity). *The qualitative model checking problem for MDPs against specification in* LTL_{D} *and* LTL$(\mathbf{F}, \mathbf{X}, \wedge)$ *is* EXPTIME*-complete.*

4.5.2 Model Checking Complexity for LTL$(\mathbf{F}, \mathbf{G}, \wedge, \vee)$

Next we show that removing nexts ($\mathbf{X}$) and untils ($\mathcal{U}$) from the logic makes the qualitative model checking problem much easier. We prove that the model checking problem of MDPs against LTL$(\mathbf{F}, \mathbf{G}, \wedge, \vee)$ is in the complexity class NP. To do so we first take a closer look at the result by Courcoubetis and Yannakakis [19] which proposed the use of LDBAs for qualitative model checking.

Proposition 4.7. *Given an MDP* $\mathcal{M}$ *and a limit-deterministic Büchi automaton* $\mathcal{A}$*, the problem of checking if there exists* $\mathfrak{S}$ *such that* $\Pr^{\mathfrak{S}}_{\mathcal{M}}(\llbracket \mathcal{A} \rrbracket) > 0$*, can be solved by taking a cross-product of* $\mathcal{M}$ *and* $\mathcal{A}$ *and checking if this product has a rechable BSCC containing a state* (s, q) *where* q *is a final state of* $\mathcal{A}$*.*

Checking for the existence of such a *final* BSCC boils down to doing a linear time graph analysis of the product. We have already seen how to transform LTL to LDBAs. The construction for $\varphi \in \text{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$ produces an exponential sized LDBA $\mathcal{A}_\varphi$, so it would seem Proposition 4.7 is not useful for proving our desired NP upper bound. The key idea we introduce here is the following: the automaton $\mathcal{A}_\varphi$ can be *split* into a disjoint union of exponentially many LDBAs each of which is polynomially large in the size of φ. In Proposition 4.7, if $\mathcal{A}$ is a disjoint union of multiple LDBAs, then the product of $\mathcal{M}$ and $\mathcal{A}$ has final BSCC if and only if the product of $\mathcal{M}$ and some individual component of $\mathcal{A}$ has a final BSCC. The NP-algorithm guesses this individual component of $\mathcal{A}$ (of polynomial size) and performs the graph analysis for the product of $\mathcal{M}$ and the indivudal component in time polynomial in both $\mathcal{M}$ and φ.

In order to understand the *splitting* of $\mathcal{A}_\varphi$ we recall the core idea behind the construction of $\mathcal{A}_\varphi$ for $\varphi \in \text{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$. For each $\mathbf{F}$ or $\mathbf{G}$ subformula ψ of φ, the automaton keeps

track of *how often* ψ *is true*, which is one of three things:

1. ψ is always true
2. ψ is true at some point but not always
3. ψ is never true

This yields a tri-partition, $\pi = \langle\, \alpha(\pi) \,|\, \beta(\pi) \,|\, \gamma(\pi) \,\rangle$, of all the $\mathbf{F}, \mathbf{G}$ subformulae of φ. If a $\mathbf{F}$ subformula is in α, β or γ we take it to mean that it is never true, true at some point but not always, or always true respectively. If $\mathbf{G}$ subformula is in α, β or γ we take it to mean that it is always true, true at some point but not always, or never true respectively. With this semantics in mind we see that a subformula in α or γ should remain in α or γ respectively in the future, and a subformula in β can remain in β only for a finite time before moving to α. It turns out that, for a given input word w, correctly guessing (a) the triple π at the beginning of w and (b) the points along w at which subformulae move from β to α, enables us to check if w satisfies the original formula φ. A key observation here is that a triple at a certain point not only tells us how often a $\mathbf{F}, \mathbf{G}$ subformula is true from that point onwards in the future, but also whether or not the subformula is true at that point. In other words, a triple refines the truth of $\mathbf{F}, \mathbf{G}$ formulae. This observation is used to inductively check that the guessed triple is correct at every point. We encourage the reader to go back and refer Section 4.1 for a detailed account of the construction. Recall that the state of the automaton for φ is of the form (π, k) where:

- π is a triple reflecting how often the $\mathbf{F}, \mathbf{G}$ subformulae are true on the remaining input.
- k is an integer counter no larger than $|\varphi|$, which is updated deterministically.

The transitions of the automaton allow moving from a state with $\pi = \langle\, \alpha \,|\, \beta \,|\, \gamma \,\rangle$ to a state with $\pi' = \langle\, \alpha' \,|\, \beta' \,|\, \gamma' \,\rangle$ only if $\alpha \subseteq \alpha'$, $\beta' \subseteq \beta$ and $\gamma = \gamma'$ ($\pi \sqsubseteq \pi'$ for short) in accordance with the semantics we associate with the triple. That is $\pi \sqsubseteq \pi'$ is a necessary condition for a transition to move from π to π', i.e., subformulae in β are allowed to move α while the remaining stay put. In order to *split* this automaton into smaller components as anticipated earlier, we add restrictions to the order in which the formulae in β are moved to α. First, let us

fix an initial triple $\pi = \langle\, \alpha_0 \,|\, \beta_0 \,|\, \gamma_0 \,\rangle$. Given π, consider a ranking function $\rho : \beta_0 \to \mathbb{N}$ whose range is allowed to be any consecutive set of positive integers starting from 1, i.e. $\{1, \ldots, n\}$. Given π and ρ we are going to define a component $\mathcal{A}_{(\pi,\rho)}$ of the original automaton $\mathcal{A}_\varphi$. We define the space of possible triples $\pi_i = \langle\, \alpha_i \,|\, \beta_i \,|\, \gamma_i \,\rangle$ for $i \in \{0, 1, \ldots, n\}$ as follows:

$$\alpha_i = \{\psi \in \beta_0 \mid f(\psi) \leq i\} \cup \alpha_0 \tag{4.5}$$

$$\beta_i = \{\psi \in \beta_0 \mid f(\psi) > i\} \tag{4.6}$$

$$\gamma_i = \gamma_0 \tag{4.7}$$

The states of $\mathcal{A}_{(\pi,\rho)}$ are those states of $\mathcal{A}_\varphi$ where the triple is restricted to be some π_i as defined above. A transition τ, say $(\pi_i, m) \xrightarrow{\sigma} (\pi_j, n)$, is allowed in $\mathcal{A}_{(\pi,\rho)}$ iff τ is a valid transition in $\mathcal{A}_\varphi$ and either $j = i$ or $j = i + 1$. In $\mathcal{A}_{(\pi,\rho)}$ a transition is allowed to either keep the triple unchanged (when $j = i$), or move only the formulae mapped to $i + 1$ from β to α (when $j = i + 1$). Thus the ranking function ρ restricts the order in which the subformulae move from β to α. A subformula with smaller rank is moved earlier compared to one with a larger rank. Note that two or more formulae can be mapped to the same number, which means those formulae are moved simultaneously. The initial state of $\mathcal{A}_{(\pi,\rho)}$ is defined to be $(\pi_0, 0)$ and the state $(\pi_n, 0)$ is marked as the only final state. Note that the size of the automaton $\mathcal{A}_{(\pi,\rho)}$ is $n + |\gamma_0|$ which is linear in $|\varphi|$. The number of different (π, ρ) is exponential in φ, hence there can be exponentially many different individual components.

What remains to be seen is that the disjoint union of these components $\biguplus \mathcal{A}_{(\pi,\rho)}$ accepts exactly the same language as $\mathcal{A}_\varphi$. Since $\mathcal{A}_{(\pi,\rho)}$ is a projection of $\mathcal{A}_\varphi$ it is the case that $[\![\mathcal{A}_{(\pi,\rho)}]\!] \subseteq [\![\mathcal{A}_\varphi]\!]$ and so $[\![\uplus \mathcal{A}_{(\pi,\rho)}]\!] \subseteq [\![\mathcal{A}_\varphi]\!]$. To see the other direction consider any word w accepted by $\mathcal{A}_\varphi$, and let $(\pi_0, k_0), (\pi_1, k_1), \ldots$ be an accepting run for w on $\mathcal{A}_\varphi$ with $\pi_i = \langle\, \alpha_i \,|\, \beta_i \,|\, \gamma_i \,\rangle$. From the construction of $\mathcal{A}_\varphi$ we know that $\pi_0 \sqsubseteq \pi_1 \sqsubseteq \pi_2 \cdots$. Identify all the positions $j_1 < j_2 < \cdots < j_n$ where the triple changes, i.e.,

$$(\pi_0 = \pi_1 \cdots = \pi_{j_1}) \sqsubset (\pi_{j_1+1} = \cdots = \pi_{j_2}) \sqsubset (\pi_{j_2+1} = \cdots = \pi_{j_3}) \sqsubset (\cdots) \sqsubset (\pi_{j_n} = \cdots \tag{4.8}$$

Here j_i is the i^{th} time the triple changes, n being the last. Now we consider the automaton $\mathcal{A}_{(\pi_0,\rho)}$ where $\rho(\psi) \stackrel{\text{def}}{=} i$ if ψ moves from β to α at position j_i, i.e., $\psi \in \beta_{j_i}$ and $\psi \in \alpha_{j_i+1}$. Observe that the above accepting run is also an accepting run of $\mathcal{A}_{(\pi,\rho)}$ on the word w. This gives us $[\![\mathcal{A}_\varphi]\!] \subseteq [\![\uplus \mathcal{A}_{(\pi,\rho)}]\!]$.

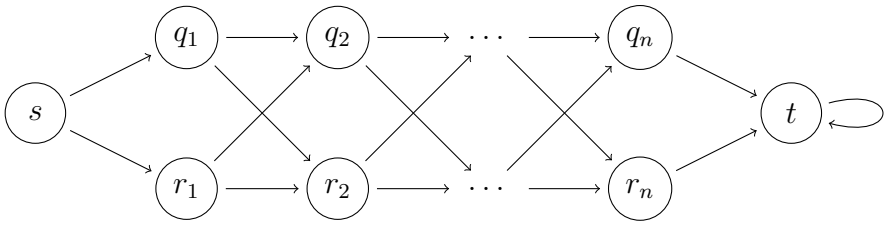


Figure 4.4: Markov chain reduction for a boolean formula having n variables.

Thus we have succesfully split $\mathcal{A}_\varphi$ into exponentially many individual components of linear size. The index (π, ρ) for any component requires only polynomially many bits to represent. This combined with our earlier observation of using Proposition 4.7 for the disjoint union gives us the NP-algorithm for qualitative model checking against LTL($\mathbf{F}, \mathbf{G}, \wedge, \vee$).

NP-hardness: Next we show that the qualitative model checking problem against the fragment LTL($\mathbf{F}, \wedge$) is NP-hard. We show this lower bound not just for MDPs but for Markov chains. We reduce the problem of boolean satisfiability to qualitative model checking of Markov Chains. Consider a boolean formula ϕ n negation normal form which consits of variables $x_1, \ldots, x_n$. We construct a Markov chain $\mathcal{M}_n$ as shown in Figure 4.4, which has start state s, a sink state t, and states $q_1, \ldots, q_n$ and $r_1, \ldots, r_n$ where n is the number of variables in ϕ. s goes to q_1 or r_1 with half probablity each. Each q_i, r_i (for $i < n$) has transitions to q_{i+1}, r_{i+1} with half probability each, and q_n, r_n proceed to t and remain there with probability 1. Next we fix the set of propositions $\{y_1, \ldots, y_n, z_1, \ldots, z_n\}$ which we will use to label the Markov chain. Each y_i is assigned to be true at and only at q_i and each z_i is set to true at and only at r_i. Next we transform the formula ϕ into a LTL($\mathbf{F}, \wedge$) formula ϕ' by replacing the positive literals x_i with $\mathbf{F}y_i$ and the negative literals $\neg x_i$ with $\mathbf{F}z_i$. For example $(x_1 \wedge \neg x_2) \vee x_3$ would become $(\mathbf{F}y_1 \wedge \mathbf{F}z_2) \vee \mathbf{F}y_3$. Now the claim is that ϕ is satisfiable if and only if ϕ' has a non-zero probability of being satisfied in $\mathcal{M}_n$. There is a one to one correspondence between assignments to variables in ϕ and paths from s to t in $\mathcal{M}_n$ such that an assignment satisfies ϕ iff the trace generated by the corresponding path satisfies ϕ'. Every path from s to t has a non-zero probability of occuring, and therefore ϕ being satisfiable is equivalent to ϕ' being satisfied in $\mathcal{M}_n$ with non-zero probability. This completes the reduction showing that qualitative model checking is NP-hard for LTL($\mathbf{F}, \wedge$).

LTL Fragment	Qualitative Model checking
LTL($\mathbf{F}, \wedge$)	NP-complete
LTL($\mathbf{F}, \mathbf{G}, \wedge, \vee$)	
LTL($\mathbf{F}, \mathbf{X}, \wedge$)	EXPTIME-complete
LTL_D	

Table 4.2: Summary of results: qualitative model checking complexity of MDPs against various fragments.

4.6 Conclusion

In this chapter we considered the qualitative model checking problem for MDPs against LTL specifications. We used the automata theoretic approach to solving the problem, which uses limit deterministic automata. We showed efficient constructions of such automata which improve upon existing constructions by an exponential factor. This results in improving the upper bound from 2EXPTIME to EXPTIME for a large class of properties, namely LTL_D. We also showed that the problem is EXPTIME-complete for LTL_D (and LTL($\mathbf{F}, \mathbf{X}, \wedge$)). We also showed that the automata we obtain for LTL($\mathbf{F}, \mathbf{G}, \wedge, \vee$), using the same translation, can be appropriately split so that they can be used in an NP algorithm for qualitative model checking, and we showed that is the best we can achieve by proving matching lower bound. These results are summarized in Table 4.2. There are some questions that still remain open. Is it possible to have an exponential sized limit deterministic translation for a logic bigger than LTL_D? This is an open question. But we do know from [52] that there is an double exponential lower bound for the full logic.

Chapter 5

Quantitative Model Checking

In this chapter we present new upper bounds for quantitative model checking of MDP for various fragments of LTL. We also prove matching lower bounds for all these fragments. The fragments we consider will be restricting the use of temporal operators to **G**, **F** and **X**.

The upper bounds we prove follow the automata theoretic approach but in a manner which differs from what we have seen previously. Here we rely on being able to construct Büchi automaton for the LTL formula, but instead of explicitly constructing the automaton from the given formula, we construct parts (states and transitions) of the automaton implicitly. The algorithm analyzes the product of the MDP and the automaton, without constructing the entire product, but constructing it on a need-to-know basis. This analysis calculates probabilities of *repeatedly reaching* a set of states in an MDP. The algorithm for doing this calculation is the core technical result that is used in all the upper bounds in this chapter. The lower bounds for these problems are obtained by adapting the lower bounds for solving 2-player games with LTL objectives.

5.1 Basic Definitions

We recall some of the notational conventions and introduce some new ones for this chapter. Given a set S, we use S^* denote the set of all finite sequences (finite words) of elements from S, and S^+ to denote all non-empty finite sequences over S. The length of a finite word u is denoted by $|u|$. We use S^ω to denote all infinite sequences over S. Given a (finite or infinite) word, we use u_i to denote i^{th} (index starting at 0) symbol in the sequence u, $u[i]$ to denote the prefix $u_0 u_1 \dots u_{i-1}$ of length i, and $u(i)$ to denote the suffix $u_i u_{i+1} \dots$ starting at index i. For $u \in S^+$, we use $\langle u \rangle$ to denote the last element in the sequence u. Given an

infinite word u, we use $\mathsf{inf}(u)$ to denote elements of S that appear infinitely often in u. We use $S^!$ to denote words in S^+ with distinct elements. The binary relations $<, \leq$ on S^* denote the prefix relations: $u < v$ iff u is a proper prefix of v. We have $u \leq v$ iff $u < v$ or $u = v$. We use $\lessdot$ to denote the covering relation of prefixes, i.e., $u \lessdot v$ iff $v = ua$ for some $a \in S$. A set $U \subseteq S^*$ is said to be closed under prefixes iff every prefix (including the empty word ϵ) of a word in U is also in U.

Definition 5.1. *A* prefix tree *on a set S is any V such that $V \subseteq S^*$, the set of vertices, is closed under prefixes. A vertex $v \in V$ is called a* leaf *if there is no $u \in V$ for which $v < u$. The set of all leaf vertices (leaves) of V is denoted by $\mathsf{Leaf}(V)$ and the set of all non-leaf vertices $\neq \epsilon$ are called inner vertices, denoted by $\mathsf{Inner}(V)$. A prefix tree is called infinite if V is infinite.*

In this chapter much of our focus will be on a particular temporal property called *repeated reachability*. For a set of states $B \subseteq Q$ we use the LTL-like notation $\mathbf{GF}B$ to denote paths that visit some state in B infinitely often, i.e., $\{\pi \in Paths(M) \mid \mathsf{inf}(\pi) \cap B \neq \emptyset\}$. For a Markov Chain M, the computation of $\Pr_M(\mathbf{GF}B)$ requires familiarity with the structure of the underlying graph of M. A set of vertices of a directed graph are called strongly connected if every pair of vertices have paths to each other. A Strongly Connected Component (SCC) is a set of vertices S that is maximally strongly connected, i.e., no superset of S is strongly connected. The SCCs of a graph induces a directed acyclic graph where the vertices are the SCCs and there is an edge from one SCC to another if there is an edge going from a vertex in the first to a vertex in the second. A SCC is called bottom (BSCC) if there is no other SCC that it can reach. Intuitively, a path of Markov chain almost certainly ends up in one of the BSCCs and visits each of the vertices in that BSCC infinitely often. In order to compute $\Pr_M(\mathbf{GF}B)$ it suffices to compute the probability of reaching BSCCs that have at least one state from B. We will be building upon these ideas in our proofs.

For a MDP $\mathcal{M}$ recall that a scheduler $\mathfrak{S}$ induces a Markov chain $\mathcal{M}_\mathfrak{S}$. A labeling for $\mathcal{M}$ using propositions AP can be extended to $\mathcal{M}_\mathfrak{S}$ which can then be used to define the probability $\Pr^\mathfrak{S}_\mathcal{M}([\![\varphi]\!])$ for a LTL formula φ over AP. The problem we are interested in this chapter is the following:

Definition 5.2. *The* quantitative *model checking problem for LTL is to decide if there exists a scheduler* $\mathfrak{S}$ *such that* $\Pr^{\mathfrak{S}}_{\mathcal{M}}([\![\varphi]\!]) \geq \theta$ *given MDP* $\mathcal{M}$*,* $\varphi \in \mathrm{LTL}$*, and* $\theta \in [0,1]$ *as inputs.*

We say two schedulers for $\mathcal{M}$, say $\mathfrak{S}_1$ and $\mathfrak{S}_2$, are equivalent, denoted by $\mathfrak{S}_1 \sim \mathfrak{S}_2$, when $Paths_f(\mathcal{M}_{\mathfrak{S}_1}) = Paths_f(\mathcal{M}_{\mathfrak{S}_2})$, and $\mathfrak{S}_1(u) = \mathfrak{S}_2(u)$ for every $u \in Paths_f(\mathcal{M}_{\mathfrak{S}_1})$. Schedulers that are equivalent yield Markov chains whose reachable portions are isomorphic. All equivalent schedulers for a MDP can be viewed as a tree obtained from "unfolding" the MDP under the scheduler. We define prefix trees associated with an MDP and then see how they relate to schedulers.

Definition 5.3. *Given a MDP* $\mathcal{M} = (Q, Act, \Delta, \mu_0)$*, a* $\mathcal{M}$-labeled prefix tree (V, λ)*, is one where* V *is a prefix tree on* Q*, and* $\lambda : \mathsf{Inner}(V) \to Act$ *is a labeling such that* $\forall u \in \mathsf{Inner}(V), q \in Q : uq \in V$ *iff* $\Delta(\langle u \rangle, \lambda(u), q) > 0$*; and* $\forall q \in Q : q \in V$ *iff* $\mu_0(q) > 0$*.*

Let $\mathcal{S}$ denote the set of all schedulers, and $\mathcal{S}/\!\sim_{\mathcal{M}}$ denote the equivalence classes induced by the $\sim_{\mathcal{M}}$ relation. The proposition below captures our observation about how equivalent schedulers of $\mathcal{M}$ can be identified by their the infinite $\mathcal{M}$-labeled prefix tree obtained from unfolding $\mathcal{M}$ on those schedulers.

Proposition 5.1. *Given MDP* $\mathcal{M} = (Q, Act, \Delta, \mu_0)$ *there is a one to one correspondence between* $\mathcal{S}/\!\sim_{\mathcal{M}}$ *and infinite* $\mathcal{M}$*-labeled prefix trees* (V, λ)*.*

5.2 Upper Bounds

In this section we present an algorithm for deciding the quantitative model checking problem when the property of interest is repeated reachability. The salient feature of the algorithm is that its space requirements are polynomial in the *diameter* but logarithmic in the graph size. Here, diameter refers to the length of the longest simple path in the graph. We exploit this algorithm to show upper bounds on quantitative model checking for wider range of properties using the automata-theoretic approach. First, we look at two class of schedulers called *memoryless* and *depth-bounded* which are useful in reasoning about the probability of repeated reachability.

Definition 5.4. *A memoryless scheduler* $\mathfrak{S}$ *is one where* $\mathfrak{S}(u) = \mathfrak{S}(v)$ *if* $\langle u \rangle = \langle v \rangle$.

A memoryless scheduler uses only knowledge about the latest state to decide which action it is going to pick. For a finite execution u, $\langle u \rangle$ represents the latest state and hence the action $\mathfrak{S}(u)$ is only dependent on $\langle u \rangle$.

Definition 5.5. *A depth-bounded scheduler* $\mathfrak{S} : Q^+ \to Act$ *is one such that*

$$\forall v \in Q^*, u \in Q^!, m \in \{1, \ldots, |u|\} : \mathfrak{S}(uu_m v) = \mathfrak{S}(u[m]v).$$

A depth-bounded scheduler can take history into consideration to make its decision, but it is only allowed to remember a bounded subsequence of the history. More precisely: at any point during the execution, the scheduler will remember a subsequence $u \in Q^!$ of the sequence of states visited so far, and either truncate or extend it depending upon whether the next state is in u or not in u. If the next state, say q, is not in u, then the new subsequence remembered is uq. Otherwise, the new subsequence is $u[m]$ where m denotes the index where q appears in u. The scheduler then chooses its action solely based on this subsequence. Note that the process of extension/truncation maintains the last state of the subsequence as the last state visited in the execution. From this it follows that a memoryless scheduler is a special case of the depth-bounded scheduler where the decision depends only on $\langle u \rangle$.

Proposition 5.2. *Every memoryless scheduler is a depth-bounded scheduler.*

Next, we define special kinds of prefix trees that correspond to depth-bounded schedulers. Let $\mathcal{D}$ denote all depth-bounded schedulers.

Definition 5.6. *A prefix tree* V *on* S *is called* depth-bounded *if* V *is finite,* $\mathsf{Inner}(V) \subseteq S^!$ *and* $\mathsf{Leaf}(V) \cap S^! = \emptyset$.

Next, analogous to Proposition 5.1, we observe how equivalent depth-bounded schedulers for $\mathcal{M}$ can be identified by $\mathcal{M}$-labeled depth-bounded prefix trees. Let $\mathcal{D}$ denote all depth-bounded schedulers, and $\mathcal{D}/\!\sim_{\mathcal{M}}$ denote the equivalence classes induced by the $\sim_{\mathcal{M}}$ relation.

Proposition 5.3. *Given MDP* $\mathcal{M} = (Q, Act, \Delta, \mu_0)$ *there is a one to one correspondence between* $\mathcal{D}/\!\sim_{\mathcal{M}}$ *and* $\mathcal{M}$*-labeled depth-bounded prefix trees.*

Next, we look at the core technical result that helps us achieve the upper bound for doing quantitative model checking. Recall that the diameter refers to the length of the longest simple path in the MDP and numeric size refers to the space required to represent the numeric quantities (probabilities) of the transitions.

Theorem 5.1. *Given a MDP $\mathcal{M} = (Q, Act, \Delta, \mu_0)$ with diameter d, graph size n and numeric size k, and given a set of states $B \subseteq Q$, the problem of deciding if $\exists \mathfrak{S} : \Pr^{\mathfrak{S}}_{\mathcal{M}}(\mathbf{GF}B) \geq \theta$ can be solved in non-deterministic space $O(d^2 \cdot (\log(n) + k))$.*

The core of the algorithm is to guess a scheduler $\mathfrak{S}$ and attempt to compute $\Pr^{\mathfrak{S}}_{\mathcal{M}}(\mathbf{GF}B)$ and compare it to θ. Recall that memoryless schedulers suffice for attaining the maximal probability of repeatedly reaching a set of states. Guessing a memoryless scheduler requires n bits of space, which does not meet our space requirement. But we know every memoryless scheduler is also a depth-bounded scheduler (Proposition 5.2), so it is suffices to look for a depth-bounded scheduler $\mathfrak{S}$. We use Proposition 5.3 to guess the $\mathcal{M}$-labeled depth-bounded prefix tree $T(\mathfrak{S}) = (V, \lambda)$. Guessing the entire tree at once would require excess space, instead we guess the tree, using depth first strategy (DFS) in a path-by-path manner. Observe that any path to a vertex in the depth-bounded tree has to be a simple path in $\mathcal{M}$ and hence bounded by the diameter d. Therefore storing a path and its labels requires only $d \cdot \log(n)$ bits of space. What remains to be seen is how to compute the required probability $\Pr^{\mathfrak{S}}_{\mathcal{M}}(\mathbf{GF}B)$ as we guess and explore the tree. The Markov chain $\mathcal{M}_{\mathfrak{S}}$ induced by a depth-bounded scheduler $\mathfrak{S}$ can be shown to be probabilistic bisimulation equivalent to Markov chain $\mathcal{M}_{T(\mathfrak{S})}$, which is obtained from $T(\mathfrak{S})$ as follows: $\mathcal{M}_{T(\mathfrak{S})} = (\mathsf{Inner}(V), \delta, \mu_0)$ where

$$\delta(u, v) \stackrel{\text{def}}{=} \begin{cases} \Delta(\langle u \rangle, \lambda(u), \langle v \rangle) & \text{if } (u \lessdot v) \text{ OR } (v \leq u \text{ and } u\langle v \rangle \in \mathsf{Leaf}(V)) \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

We classify the edges of the above Markov chain as: (i) *forward edges* (u, v) where $u \lessdot v$, (ii) *back edges* (u, v) where $v \leq u$.

The probability of repeatedly reaching B in $\mathcal{M}_{\mathfrak{S}}$ equals the probability of repeatedly reaching B' in $\mathcal{M}_{T(\mathfrak{S})}$ where $B' = \{v \in \mathsf{Inner}(V) \mid \langle v \rangle \in B\}$. Computing repeated reachability in Markov chains boils down to computing probability of reaching BSCCs that contain

at least one of the given states. In the remainder of this section we see how to do this in the DFS that explores the tree $T(\mathfrak{S})$.

Index of a SCC: For a SCC of $\mathcal{M}_{T(\mathfrak{S})}$ define an *index* vertex w as one for which there is no other vertex w' in the SCC such that $w' < w$. The first observation is that there is a unique index vertex for a SCC. For contradiction assume there are two indices $u \neq v$ for a SCC. By definition of SCC there is a simple path from u to v. If this path does not consist of any back edges then clearly $u < v$ contradicting the fact that v is an index. If the path consists of back edges consider the first back edge in the path, say (u', v'). Now v' cannot be on the path from u to u', due to the fact that nodes cannot be repeated on a simple path (otherwise v' would be visited twice from u to v). So $v' < u'$ (because back edges lead to a prefix/ancestor) and $u \not< v'$. This means $v' < u$ because two ancestors of any node in a tree are always directly related (Any prefix of u' which is not a suffix of u has to be a prefix of u). Since there is a path from u to v' (as observed), and a path from v' to u owing to the fact that $v' < u$, we get that v' is included in the SCC. This contradicts u being an index. This proves the uniqueness of an index node. In our algorithm we guess which nodes in the tree are index nodes and compute the probability of reaching every index node. The probability of reaching a BSCC is simply the probability of reaching the index of that BSCC. Also note that every node $V \cap Q$ is an index node, as there is no vertex in $\mathsf{Inner}(V)$ that precedes such vertices.

Parent-Child relationship: In order to compute the probability of reaching an index node in an inductive fashion we identify the parent-child relationship between indices. An index node v is called the *child* of an index node u if $u < v$ and there is no index node w such that $u < w < v$. u is called the *parent* of v, if v is the child of u. Note that every index has a unique parent except for the nodes in $V \cap Q$ which have no parent, so we call $V \cap Q$ the root nodes. For an index node u let $C(u)$ denote all the children of u. Given a node u, let p_u denote the probability of reaching u. Given a node uv with $u, v \neq \epsilon$, let $q_{u,v}$ denote the

probability of moving from u to uv along the path of forward eges from u to uv in $\mathcal{M}_{T(\mathfrak{S})}$:

$$q_{u,v} \stackrel{\text{def}}{=} \prod_{i=0}^{|v|-1} \delta(u.v[i], u.v[i+1]) \tag{5.2}$$

The Proposition below formulates how we can calculate the probability of reaching an index by using the reachability probability of its parent.

Proposition 5.4. *For a node $uv \in C(u)$, the probability p_{uv} of reaching uv is given by $(p_u \cdot q_{u,v})/s_u$, where $s_u \stackrel{\text{def}}{=} \sum_{w \in C(u)} q_{u,w}$ is called the normalizing factor of u.*

In order to use the above formula for the computation of p_{uv} we need to know the normalizing factor s_u of the parent node u. We guess this quantity s_u associated with each node u that is an index, and store it along with u on the depth-first stack. Now, let us see how these can be used to compute reachability probabilities. For an index node uv whose parent is u, assume p_u is already computed and stored. The parent of a node can be identified by looking at the latest node before u in the stack that is an index node. For the root nodes r, $p_r = \mu_0(r)$. Now p_{uv} can be computed according to Proposition 5.4 using:

- p_u which is already computed and stored on the stack when u was first encountered
- s_u which is guessed and stored on the stack when u was first encountered
- $q_{u,v}$ which can be computed by looking at the path from u to uv on the depth-first stack.

Next, to compute the probabilities of reaching the BSCCs that have a state from B' in them, we observe that an index node u corresponds to a BSCC iff the normalizing factor $q_u = 0$, implying that it has no children. For such a state u we mark it as *final* as soon as a descendant uv is encountered where $\langle uv \rangle \in B$. Once all the descendants of u are explored we check if it is final, and if so we add the probability of reaching it, p_u, to a running total. The total value at the end of the DFS exploration is the required probability $\Pr^{\mathfrak{S}}_{\mathcal{M}}(\mathbf{GF}B)$.

Confirming guesses: In the computation described above we have guessed two things for every node u:

- if u is an index or not
- the normalizing factor s_u, whenever u is an index

In order to check that our guess regarding u being an index is correct we use the following:

Proposition 5.5. *For $T(\mathfrak{S}) = (V, \lambda)$, a node $u \in \mathsf{Inner}(V)$ is an index node of some SCC in $\mathcal{M}_{T(\mathfrak{S})}$ iff every $uv \in \mathsf{Leaf}(V)$ is such that $\langle uv \rangle = \langle uv' \rangle$ for some $uv' \in \mathsf{Inner}(V)$.*

So, when u is guessed as an index node we make sure that every leaf node descending from u points to a repetition of a state that is no earlier than u, and when u is guessed as non-index we ensure there is a leaf node descending from u pointing to a repetition of a state earlier than u. In order to check that the guess for s_u is correct, we maintain a running sum for each index node u on the stack. When a $uv \in C(u)$ is encountered we add the computed quantity $q_{u,v}$ to the running sum associated with u. When the DFS exploration for u is complete we check that the running sum equals the guess s_u.

Size of the quantities: So far in this algorithm we have not accounted for the space requirements of the quantities we calculate. Let us begin by looking at $q_{u,v}$ for parent-child indices u, v. Equation 5.2 tells us that $q_{u,v}$ is a product of transition probabilities from u to v of which there are at most d. Therefore $q_{u,v}$ requires $d{\cdot}k$ bits to store since each transition probability has no more than k bits. Next, the normalizing factor s_u for an index u is the sum of $q_{u,v}$ where uv is a child of u. Note that the number of children for any index is bounded by the total number of nodes in the tree which is at most n^d. So each s_u, the sum of n^d quantities ($q_{u,v}$) each of size $d{\cdot}k$ requires only $d{\cdot}(\log(n)+k)$ bits. For p_u observe that it can be written as a product of $q_{u',v'}/s_{u'}$ corresponding to the ancestor indices of u of which there are at most d. Hence each p_u takes $O(d^2 \cdot (\log(n) + k))$ bits. The sum of p_u for u that are final BSCCs of which there are no more than n^d, will increase the bits required by $d{\cdot}\log(n)$. So the total space required remains $O(d^2 \cdot (\log(n)+k))$. This finishes up the proof for Theorem 5.1.

Upper bounds for LTL fragments

We are now ready to present all our upper bounds for the quantitative verification problem for different LTL fragments. Our results rely on the automata theoretic approach that solves the quantitative verification problem by constructing a deterministic automaton for the given LTL specification. We, therefore, begin by recalling results on translations of fragments of LTL to deterministic automata.

Theorem 5.2. *(Alur and La Torre [3]) The following fragments of LTL can be translated into deterministic Buchi automata with the following space and diameter bounds.*

- $\text{LTL}(\mathbf{F}, \wedge)$ *has automata of exponential size and linear diameter.*
- $\text{LTL}(\mathbf{F}, \mathbf{X}, \wedge)$ *has automata of exponential size and exponential diameter.*
- $\text{LTL}(\mathbf{F}, \wedge, \vee)$ *has automata of double exponential size and exponential diameter.*
- $\text{LTL}(\mathbf{F}, \mathbf{X}, \wedge, \vee)$ *has automata of double exponential size and exponential diameter.*
- $\text{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$ *has automata of double exponential size and double exponential diameter.*

These bounds on size and diameter are also tight.

Using these we present our upper bound results, also summarized in Table 5.1 at the end of this chapter.

Theorem 5.3. *The quantitative verification problem for MDPs against LTL specifications has the following complexity bounds for* $\mathcal{B}(\text{LTL}(\mathbf{F}, \wedge))$ *it is in PSPACE; for* $\mathcal{B}(\text{LTL}(\mathbf{F}, \wedge, \vee))$ *it is in EXPSPACE; for* $\mathcal{B}(\text{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee))$ *it is in 2EXPTIME; for* $\mathcal{B}(\text{LTL}(\mathbf{F}, \mathbf{X}, \wedge))$ *it is in EXPTIME; for* $\mathcal{B}(\text{LTL}(\mathbf{F}, \mathbf{X}, \wedge, \vee))$ *it is in EXPSPACE; for* $\mathcal{B}(\text{LTL}(\mathbf{F}, \mathbf{G}, \mathbf{X}, \wedge, \vee))$ *it is in 2EXPTIME.*

Proof Sketch: Recall that in the automata theoretic approach to quantitative verification of MDPs, the LTL specification φ is translated into a deterministic automaton $\mathcal{A}$, and then the cross product of $\mathcal{A}$ with the MDP $\mathcal{M}$ is analyzed. When the automaton $\mathcal{A}$ is Buchi, the

analysis involves solving the repeated reachability problem on the cross product MDP. The algorithm of [58, 19] runs in time that is polynomial in the size of the cross product. Given the results on the size of deterministic Buchi automata mentioned in Theorem 5.2, we immediately get the complexity bounds for LTL($\mathbf{F}, \mathbf{G}, \wedge, \vee$), LTL($\mathbf{F}, \mathbf{X}, \wedge$) and LTL($\mathbf{F}, \mathbf{G}, \mathbf{X}, \wedge, \vee$). For the other upper bounds, we follow a similar approach, but we construct the product of $\mathcal{M}$ and $\mathcal{A}$ on the fly. We exploit the fact that the Buchi automata constructions for LTL formulae have an amenable representation that allows one to guess its states and check the transition relation just from the knowing the formula. This allows us to apply Theorem 5.1 to the implicit product whose diameter is the product of the diameters of $\mathcal{M}$ and $\mathcal{A}$. Given the bounds on the diameter of the deterministic Buchi automata mentioned in Theorem 5.2, and using Theorem 5.1, we obtain the complexity bounds for the remaining fragments.

5.3 Lower Bounds

In this section we prove matching lower bounds for the upper bounds established in Theorem 5.3. The lower bounds essentially follow from lower bounds established in [3, 4] for 2-player games. The reason for this observation is that games constructed in the lower bound reductions in [3, 4] have a special property that enable their lifting to the quantitative verification problem for MDPs. We begin this section by identifying this property, and showing how it helps transfer complexity bounds to the quantitative verification case.

Recall that a two player game is played on a graph $G = (V, E)$, where the set of vertices V is partitioned into two sets $V_\exists$ which belong to $\exists$-player, and $V_\forall$ which belong to $\forall$-player. At any given time, the play is at some vertex u of graph G. Player P ($P \in \{\exists, \forall\}$) plays from u if u V_P , by picking the target of some outgoing edge from u. Starting from an initial vertex u_0, a play is the infinite sequence of vertices visited as the players choose edges on their turn. Given an objective described by LTL formula φ, we say a play π is winning for $\exists$-player if π satisfies φ; otherwise the play is said to be winning for the $\forall$-player. We now identify a special class of games that we call finitely winnable.

Definition 5.7. *A game (G, φ) is said to* finitely-winnable *for a player P ($P \in \{\exists, \forall\}$) iff for any play π of (G, φ) that P wins, there is a prefix of π, say π', such that every play*

(according to the game graph G) that is an extension of π' is also winning for P.

The main observation about finitely winnable games is that if the $\forall$-player is replaced by a stochastic player that uniformly chooses among the available choices, then in the resulting MDP, there is a scheduler that meets objective φ with probability 1 if and only if the $\exists$-player has a winning strategy in the game.

Proposition 5.6. *Given a game (G, φ) which is finitely-winnable for the $\forall$-player, the MDP $\mathcal{M}_G$ obtained by replacing the $\forall$-player with stochastic choices is such that: the $\exists$-player has a winning strategy for (G, u_0, φ) if and only if there exists a scheduler $\mathfrak{S}$ such that $\Pr^{\mathfrak{S}}_{\mathcal{M}_G}([\![\varphi]\!]) = 1$.*

Proof. If the $\exists$-strategy has a winning strategy for (G, u_0, φ), then the strategy interpreted as scheduler for $\mathcal{M}_G$ is going to be such that all runs of that scheduler are going to satisfy φ, which implies $\Pr^{\mathfrak{S}}_{\mathcal{M}_G}([\![\varphi]\!]) = 1$. Now consider the case the where the $\forall$-player has a winning strategy for (G, u_0, φ). Here, for any strategy of the $\exists$-player there is going to be a play that is won by the $\forall$-player. Since the game is finitely-winnable for the $\forall$-player, we know there is a prefix of the play whose every extension is winning for the $\forall$-player. What this means in the MDP setting, is that for any strategy there is a finite run ρ whose every extension results in φ not being met. Since the measure associated with all extensions of ρ is non-zero (since ρ is finite), we get that any strategy loses with non-zero probability, i.e., $\Pr^{\mathfrak{S}}_{\mathcal{M}_G}([\![\varphi]\!]) < 1$ for any scheduler $\mathfrak{S}$. □

We use the above observations to obtain matching lower bounds for the quantitative verification problem.

Theorem 5.4. *The quantitative verification problem for MDPs against LTL specification has the following complexity lower bounds:*

- *for $\mathcal{B}(\mathrm{LTL}(\mathbf{F}, \wedge))$ it is PSPACE-hard*
- *for $\mathcal{B}(\mathrm{LTL}(\mathbf{F}, \wedge, \vee))$ it is EXPSPACE-hard*
- *for $\mathcal{B}(\mathrm{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee))$ it is 2EXPTIME-hard*

- *for* $\mathcal{B}(\text{LTL}(\mathbf{F}, \mathbf{X}, \wedge))$ *it is EXPTIME-hard*
- *for* $\mathcal{B}(\text{LTL}(\mathbf{F}, \mathbf{X}, \wedge, \vee))$ *it is EXPSPACE-hard*
- *for* $\mathcal{B}(\text{LTL}(\mathbf{F}, \mathbf{G}, \mathbf{X}, \wedge, \vee))$ *it is 2EXPTIME-hard.*

Proof Sketch: All the lower bounds follow from similar lower bounds established for solving 2-player games for the same LTL fragments in [3, 4]. All reductions for games essentially reduce the membership problem of a space/time bounded Alternating Turing machine (ATM) given an ATM $\mathcal{A}$ and an input w they construct a game graph G, initial state u_0, and a specification φ such that $w \in L(\mathcal{A})$ iff there exists a winning strategy for the $\exists$-player in the game (G, u_0, φ). In each of these reductions, the game (G, u_0, φ) is finitely winnable for the $\forall$-player. Thus, we can use the same reduction and Proposition 5.6 to obtain a lower bound for the quantitative verification problem for MDPs when the threshold is 1.

LTL Fragment	Quantitative Model checking
$\mathcal{B}(\text{LTL}(\mathbf{F}, \wedge))$	PSPACE-complete
$\mathcal{B}(\text{LTL}(\mathbf{F}, \mathbf{X}, \wedge))$	EXPTIME-complete
$\mathcal{B}(\text{LTL}(\mathbf{F}, \wedge, \vee))$	EXPSPACE-complete
$\mathcal{B}(\text{LTL}(\mathbf{F}, \mathbf{X}, \wedge, \vee))$	
$\mathcal{B}(\text{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee))$	2EXPTIME-complete
$\mathcal{B}(\text{LTL}(\mathbf{F}, \mathbf{G}, \mathbf{X}, \wedge, \vee))$	

Table 5.1: Summary of results: quantitative model checking complexity of MDPs against various fragments.

5.4 Conclusion

In this chapter we considered the problem of quantitative model checking of MDPs against LTL fragments. The existing approach to the problem involves a translation of the given

formula into a deterministic automaton, followed by an analysis of the cross-product of the automaton and the MDP. Therefore this approach directly depends on the size of the automaton. We proved that for certain fragments one can improve upon this automata based approach by exploiting the fact that the translations yield automata in which the diameter is asymptotically smaller than its size. Our core technical result involves a space-efficient algorithm for calculating repeated reachability probabilities which is polynomial in the diameter but logarithmic in the size of the MDP . This combined with the known results on translation of LTL fragments to deterministic automata give us our upper bounds. We also proved matching lower bounds for all the fragments we considered. In Table 5.1 we summarize the complexities of the quantitative model checking problem for different fragments.

Chapter 6

Experiments with Büchifier

We present our tool `Büchifier` (available at [1]) that implements the techniques described in this paper. `Büchifier` is the first tool to generate LDBA with provable exponential upper bounds for a large class of LTL formulae. The states (μ, ν, π, n) in our automaton described in Definition 4.7, involve $\mu, \nu \in \mathcal{B}^+(\varphi)$ which are essentially sets of sets of subformulae. We view each subformula as a different proposition. We then interpret the formulae in $\mathcal{B}^+(\varphi)$ as a Boolean function on these propositions. In `Büchifier` we represent these Boolean functions symbolically using Binary Decision Diagrams (BDD). Our overall construction follows a standard approach where we begin with an initial set of states and keep adding successors to discover the entire reachable set of states. We report the number of states, number of transitions and the number of final states for the limit deterministic automata we construct.

MDP model checkers like PRISM [39], for a long time have used the translation from LTL to deterministic Rabin automata and only recently [52] have started using limit deterministic Büchi automata. As a consequence we compare the performance of our method against `Rabinizer 3` [32] (the best known tool for translating LTL to deterministic automata) and `ltl2ldba` [52] (the only other known tool for translating LTL to LDBA). `Rabinizer 3` constructs deterministic Rabin automata with generalized Rabin pairs (DGRA). The experimental results in [22, 32] report the size of DGRA using the number of states and number of acceptance pairs of the automata; the size of each Rabin pair is, unfortunately, not reported. Since the size of Rabin pairs influences the efficiency of MDP model checking, we report it here to make a meaningful comparison. We take the size of a Rabin pair to be simply the number of transitions in it. The tool `ltl2ldba` generates transition-based generalized Büchi automata (TGBA). The experimental results in [52] report the size of the TGBA using

	Büchifier			Rabinizer 3			ltl2ldba		
	St	Tr	AC	St	Tr	AC	St	Tr	AC
$g_0(1)$	4	7	2	1	1	3	3	6	2 (1)
$g_0(2)$	12	23	5	1	1	8	5	14	12 (2)
$g_0(3)$	32	63	8	1	1	20	9	36	54 (3)
$g_1(2)$	12	21	5	1	1	8	5	13	11 (2)
$g_1(3)$	31	54	13	1	1	18	9	30	44 (3)
φ_1	5	7	3	5	13	40	7	23	12 (4)
φ_2	26	83	8	12	48	233	36	101	75 (2)
φ_3	13	25	3	16	128	64	21	140	129(2)
φ_4	17	47	7	2	4	35	9	29	31 (2)
φ_5	36	111	11	12	48	330	41	133	94 (2)
$f_0(1)$	4	7	2	2	4	2	2	4	2 (1)
$f_0(2)$	14	29	5	16	74	26	4	16	16 (2)
$f_0(3)$	44	105	13	–	–	–	8	64	96 (3)
$f_0(4)$	130	369	33	–	–	–	16	256	512(4)
$f_1(1)$	14	29	5	6	24	10	8	32	12 (1)
$f_1(2)$	130	369	33	–	–	–	64	1024	768(2)
$f_1(3)$	1050	4801	193	–	–	–	512	32768	36K(3)
$f_2(1)$	1	1	1	2	3	2	1	1	2 (2)
$f_2(2)$	5	7	3	5	13	45	6	21	9 (3)
$f_2(3)$	19	37	7	19	109	847	19	218	28 (4)
$f_2(4)$	65	175	15	167	2529	–	93	6301	75 (5)

Table 6.1: A Comparison between the sizes of automata produced by `Büchifier`, `Rabinizer 3` and `ltl2ldba` on various formulae. Column St denotes the number of states, column Tr denotes the number of transitions and column AC denotes the size of the acceptance condition. Entries marked as "–" indicate that the tool failed to construct the automaton and/or the acceptance condition due to the memory limit (1GB) being exceeded.

	Büchifier			Rabinizer 3			ltl2ldba		
	St	Tr	AC	St	Tr	AC	St	Tr	AC
$f_3(1)$	2	4	1	3	7	4	1	2	3 (2)
$f_3(2)$	10	20	4	17	91	53	14	62	28 (1)
$f_3(3)$	36	78	12	–	–	–	212	2359	953(1)
$f_3(4)$	114	288	32	–	–	–	17352	598330	167K(1)
$h(2,1)$	26	54	9	15	49	49	14	44	1(1)
$h(2,2)$	60	138	21	65	469	469	64	434	1(1)
$h(2,3)$	182	468	57	315	5119	5119	314	4892	1(1)
$h(4,1)$	80	146	36	76	250	250	75	229	1(1)
$h(4,2)$	230	464	96	990	8068	8068	989	7465	1(1)
$h(4,3)$	908	1994	348	–	–	–	–	–	–
ψ_1	35	62	9	3	6	12	3	6	8 (3)
ψ_2	7	15	3	8	39	53	2	5	18 (3)
ψ_3	29	62	8	29	116	74	62	293	27(2)
ψ_4	26	92	6	4	11	7	3	8	3(1)
ψ_5	9	58	1	5	17	9	3	9	3(1)

Table 6.1 (continued)

number of states and number of acceptance sets, and once again the size of each of these sets is not reported. Since their sizes also effect the model checking procedure we report them here. We take the size of an acceptance set to be simply the number of transitions in it. In Table 6.1 we report a head to head to comparison of `Büchifier`, `Rabinizer 3` and `ltl2ldba` on various LTL formulae.

1. The first 5 formulae are those considered in [22]; they are from the GR(1) fragment [45] of LTL. These formulae capture Boolean combination of fairness conditions for which generalized Rabin acceptance is particularly well suited. `Rabinizer 3` does well on these examples, but `Büchifier` is not far behind its competitors. The formulae are instantiations of the following templates: $g_0(j) = \wedge_{i=1}^{j}(\mathbf{GF}a_i \Rightarrow \mathbf{GF}b_i)$,

$g_1(j) = \wedge_{i=1}^{j}(\mathbf{GF}a_i \Rightarrow \mathbf{GF}a_{i+1})$.

2. The next 5 formulae are also from [22] to show how `Rabinizer 3` can effectively handle $\mathbf{X}$s. `Büchifier` has a comparable number of states and much smaller acceptance condition when compared to `Rabinizer 3` and `ltl2ldba` in all these cases. $\varphi_1 = \mathbf{G}(q \vee \mathbf{XG}p) \wedge \mathbf{G}(r \vee \mathbf{XG}\neg p)$, $\varphi_2 = (\mathbf{GF}(a \wedge \mathbf{X}^2 b) \vee \mathbf{FG}b) \wedge \mathbf{FG}(c \vee (\mathbf{X}a \wedge \mathbf{X}^2 b))$, $\varphi_3 = \mathbf{GF}(\mathbf{X}^3 a \wedge \mathbf{X}^4 b) \wedge \mathbf{GF}(b \vee \mathbf{X}c) \wedge \mathbf{GF}(c \wedge \mathbf{X}^2 a)$, $\varphi_4 = (\mathbf{GF}a \vee \mathbf{FG}b) \wedge (\mathbf{GF}c \vee \mathbf{FG}(d \vee \mathbf{X}e))$, $\varphi_5 = (\mathbf{GF}(a \wedge \mathbf{X}^2 c) \vee \mathbf{FG}b) \wedge (\mathbf{GF}c \vee \mathbf{FG}(d \vee (\mathbf{X}a \wedge \mathbf{X}^2 b)))$.

3. The next 15 formulae (4 groups) express a variety of natural properties, such as $\mathbf{G}(req \Rightarrow \mathbf{F}ack)$ which says that every request that is received is eventually acknowledged. As shown in the table in many of the cases `Rabinizer 3` runs out of memory (1GB) and fails to produce an automaton, and `ltl2ldba` fails to scale in comparison with `Büchifier`. The formulae in the table are instantiations of the following templates: $f_0(j) = \mathbf{G}(\wedge_{i=1}^{j}(a_i \Rightarrow \mathbf{F}b_i))$, $f_1(j) = \mathbf{G}(\wedge_{i=1}^{j}(a_i \Rightarrow (\mathbf{F}b_i \wedge \mathbf{F}c_i)))$, $f_2(j) = \mathbf{G}(\vee_{i=1}^{j}(a_i \wedge \mathbf{G}b_i))$, $f_3(j) = \mathbf{G}(\vee_{i=1}^{j}(a_i \wedge \mathbf{F}b_i))$.

4. The next 6 formulae expressible in LTL$\backslash GU$, contain multiple $\mathbf{X}$s and external $\mathcal{U}$s. `Büchifier` constructs smaller automata and is able to scale better than `ltl2ldba` in these cases as well. The formulae are instantiations of: $h(m,n) = (\mathbf{X}^m p)\,\mathcal{U}\,(q \vee (\wedge_{i=1}^{n}(a_i\,\mathcal{U}\,\mathbf{X}^m b_i)))$.

5. The last few examples are from outside of LTL$\backslash GU$. The first three are in $\mathrm{LTL_D}$ while the rest are outside $\mathrm{LTL_D}$. We found that `Büchifier` did better only in a few cases (like ψ_3), this is due to the multiplicative effect that the internal untils have on the size of the automaton. So there is scope for improvement and we believe there are several optimizations that can be done to reduce the size in such cases and leave it for future work. $\psi_1 = \mathbf{FG}((a \wedge \mathbf{X}^2 b \wedge \mathbf{GF}b)\,\mathcal{U}\,(\mathbf{G}(\mathbf{X}^2\neg c \vee \mathbf{X}^2(a \wedge b))))$, $\psi_2 = \mathbf{G}(\mathbf{F}\neg a \wedge \mathbf{F}(b \wedge \mathbf{X}c) \wedge \mathbf{GF}(a\,\mathcal{U}\,d))$, $\psi_3 = \mathbf{G}((\mathbf{X}^3 a)\,\mathcal{U}\,(b \vee \mathbf{G}c))$, $\psi_4 = \mathbf{G}((a\,\mathcal{U}\,b) \vee (c\,\mathcal{U}\,d))$, $\psi_5 = \mathbf{G}(a\,\mathcal{U}\,(b\,\mathcal{U}\,(c\,\mathcal{U}\,d)))$.

Chapter 7

Distribution Based Semantics

In this second part of the thesis (from here to Chapter 9) we focus on the distribution based semantics. Here, the non-determinism in MDPs is resolved by Markovian schedulers. A Markovian scheduler (which we shall refer to as a schedule) is an infinite sequence of actions. In other words these schedules choose their action purely based on the number of steps that have been taken. Such a schedule then induces a sequence of probability distribution on the states of the MDP. These distributions are then classified based on labelings. Note that the labelings here are not applied to the states (as in the execution based semantics) but to the probability distribution of states. The labelings of interest are what we call *binary* labels, which compares the probability of being in a certain set of states to a constant, to assign truth to propositions. We further focus on binary labels that are *robust*, where perturbing the constants by a small amount does not change their truth. We prove a number of decidability results for checking robustness (Chapter 8) and model checking Markov chains under robustness (Chapter 9), and contrast it with the corresponding complexity for MDPs.

We begin by illustrating a working example borrowed from [33], that motivates the use of distribution based semantics. We look at compartment models which are used in drug administration [51]. A compartment is a group of organs or tissues that maintain similar profiles of blood flow and drug concentration. These compartments have the property that the rate of flow of any drug leaving a compartment is proportional to the concentration of the drug in it. This essentially allows us to model drug concentrations as probabilities which flow from one compartment to another. Thus Markov chains emerge as a convenient formalism where the states act as compartments and the probabilities act as concentrations. But sometimes Markov chains do not suffice, because the rate of flow of concentration is not constant but is multimodal. For instance the drug disposition shows a saturation behaviour

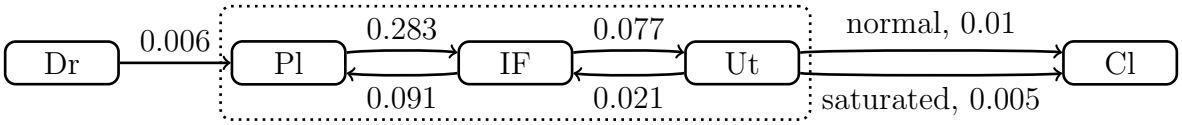


Figure 7.1: The MDP corresponding to a compartment model of Insulin$-^{131}$I

when the body has more drugs than certain enzymes. So there is a normal mode when the flow behaves in a certain way and there is a saturation mode when it behaves in another way. The switching between these modes may be out of our control or unpredictable. In such scenarios one can model the switching as a non-deterministic behaviour, and this is where MDPs become useful.

We look at a specific example of the behaviour of Insulin$-^{131}$I in the human body modeled using compartments [53] in Figure 7.1. The states Pl, IF, and Ut correspond to compartments Plasma, Interstitial fluid, and Utilization site which are present in the body (body marked by the dotted box). The probability of being in these states corresponds to the concentration of Insulin$-^{131}$I in those compartments. The probability at state Dr corresponds to the total amount of drug administerd and Cl corresponds to a sink state that clears drug away from the body. The transitions from one state to another describes the rate of flow between the corresponding compartments in one unit of time. The unit of time in the above model is 10 minutes. We have not chosen to represent transitions from a state to itself, but they can inferred from the other outgoing transitions. The only non-deterministic behaviour is present at state Ut, where the clearance of insulin from the body depends upon whether the system is in normal or saturated mode. Note that two different actions are enabled at Ut, namely normal and saturated, and both of them have the same rate of flow to IF, and differing rate of flow to Cl. This is balanced by the implicit self transitions at state Ut on each of these actions. To begin with we can put a initial amount of drug c_0 in the Dr state. By choosing the appropriate unit we can ensure $0 \leq c_0 \leq 1$, and then put the remaining $1 - c_0$ units in the state Cl.

Next we look at the requirements of this compartment model. We would like to verify properties of the following kind: no matter how the modes change, drug concentration at Ut is (i) always below a certain *toxic threshold* "*tt*", and (ii) above a given *effective threshold*

"*et*" for a certain amount of time. We might even want a stronger property that the drug concentration is always *well below* the toxic threshold and not just barely acceptable. By this we mean that there is some positive constant ϵ such that the drug concentration is below $tt - \epsilon$. This is what we shall refer to as *robustness*, which we shall define precisely in the next section. The mode changes of the system can be thought of as a "schedule" which chooses at each point whether the system is in normal mode or saturated mode. In our example a schedule is an infinite sequence of actions over the binary alphabet $\Sigma = \{normal, saturated\}$. If we are given a schedule $\sigma \in \Sigma^\omega$ and a probability distribution μ_0 describing the initial concentrations, then we can infer concentrations at every time point. Let μ_t denote the concentration at time t and $\mu_t(s)$ denote the concentration in state s at time t, then we have $\mu_{t+1} = \mu_t \cdot \sigma_t$, where σ_t is the t^{th} symbol in the sequence σ.

The requirements on the above model are given as ω-regular properties described in Section 1.2.2. We first define atomic propositions *effect* and *toxic* on the probability distribution over the states. A probability distribution μ assigns *effect* to be true if $\mu(Ut) > et$ and false otherwise. So *effect* is said to be true at time t iff $\mu_t(Ut) > et$. Similarly μ assigns *toxic* to be true if $\mu(Ut) \geq tt$. Now we can describe the temporal property, "always below toxic threshold" in LTL as $\mathbf{G}(\neg toxic)$. The property "always well below toxic threshold" would be true if $\mathbf{G}(\neg toxic)$ holds true and the proposition *toxic* is robust. For the second requirement if it is the case that we require insulin concentration to be above *et* for 3 consecutive units of time then we can say this using $\mathbf{F}(effect \wedge (\mathbf{X}effect) \wedge (\mathbf{XX}effect))$. For this example we have used LTL to specify the requirements but more generally we can use Büchi automata over assignments to appropriate propositions to do so. In this part of the thesis we shall explore questions regarding the complexity of determining robustness of a proposition and the complexity of model checking properties described using robust propositions.

7.1 Basic Definitions

We review some of the terminology and basic results required to proceed with the results in this part of the thesis.

7.1.1 Distributions and Stochastic Matrices.

A *probability distribution* over a finite set S is a function $\mu : S \to [0,1]$ such that $\sum_{s\in S} \mu(s) = 1$. We use $Dist(S)$ to denote the set of all distributions over the set S. Let $s_1, s_2, \ldots, s_n$ be an enumeration of a finite set S. A collection of distributions $X \subseteq Dist(S)$ is termed *binary* if there are $a_1, a_2, \ldots, a_n \in \{0,1\}$, $b \in [0,1] \cap \mathbb{Q}$ such that $\mu \in X$ iff $\sum_i a_i \mu(s_i) > b$. For a set $S' \subseteq S$, we write $\mu(S') = \sum_{i\in S'} \mu(i)$. For any two distributions $\mu, \nu \in dist(S)$ the distance between them is defined as

$$\boldsymbol{d}(\mu,\nu) = \sum_{i\in S} \frac{|\mu(i) - \nu(i)|}{2} = \max_{S'\subseteq S} |\mu(S') - \nu(S')|. \tag{7.1}$$

The distance $\boldsymbol{d}$ is a metric.

A stochastic matrix δ is a square matrix with non-negative entries such that each row of the matrix sums up to one. This distance between $n \times n$ matrices δ_1,δ_2 is defined as:

$$\boldsymbol{d}(\delta_1,\delta_2) = \max_i \sum_j |\delta_1(i,j) - \delta_2(i,j)|. \tag{7.2}$$

We use $\delta(;,j)$ to represent the jth column of the matrix δ. We use $\sigma_t(\delta)$ to denote the sequence of matrices $\delta^t, \delta^{2t}, \delta^{3t}, \ldots$ and use $\widehat{\delta^t}$ to denote the limit $\lim_{r\to\infty} \delta^{rt}$ if it exists. When clear from the context, we shall drop δ from $\sigma_t(\delta)$ and just write σ_t. A stochastic matrix δ is called *positive* if all of its entries are strictly positive.

A stochastic matrix of dimension $n \times n$ can be represented as a directed graph with n vertices, and an edge from i to j if $\delta(i,j) > 0$. A maximally strongly connected component of the graph is called a *Bottom Strongly Connected Component* (BSCC) if it has no outgoing edges. A *transient state* is a state which is not in a BSCC, and a *terminal state* is one which is within a BSCC. δ is said to be *irreducible* if it has only one BSCC and no transient states. The collection of all BSCCs of a δ will be represented by $\mathcal{C}_\delta$. The set of all transient states of δ will be denoted by T_δ. Lower case c_δ will be used to denote individual BSCCs. When clear from the context, we shall drop the subscript δ.

The *period* of a vertex is defined as the g.c.d (greatest common divisor) of all the cycle lengths going through the vertex. For a SCC, the periods of all the vertices in that component will be the same and will be defined as the period of that component. A component is called

aperiodic if its period is 1. δ is said to be *aperiodic* if all vertices have period 1. The *ultimate period* of δ is the l.c.m (least common multiple) of the periods of its BSCCs. Since BSCCs and their related periods can be computed in polynomial time we have the following:

Proposition 7.1. *The ultimate period of a $n \times n$ matrix δ can be computed in polynomial time and is a number with $O(n \log n)$ bits.*

The following Lemma is proved in [50]

Lemma 7.1. *For any $n \times n$ stochastic matrix δ, if δ is an aperiodic and irreducible stochastic matrix then δ^{n^2} is positive.*

A stochastic matrix γ with dimensions $n \times n$ is called a *contraction map with contracting factor* $\alpha < 1$ if for all distributions μ and ν of dimension n it is the case that $\boldsymbol{d}(\mu\gamma, \nu\gamma) < \alpha \boldsymbol{d}(\mu, \nu)$.

Proposition 7.2. *For any $n \times n$ stochastic matrix δ if δ is positive then δ is contracting with contracting factor $1 - n \min_{i,j} \delta(i, j)$.*

7.1.2 Schedules, Labeling and Model Checking

Schedule. Given an MDP $\mathcal{M} = (Q, Act, \Delta, \mu_0)$, a *schedule* σ for $\mathcal{M}$ is an infinite sequence of actions from Act. The set of all schedules is represent as Act^ω. Given schedule σ the *path* induced by σ, is the sequence of distributions $path(\sigma) = \mu_0\mu_1\mu_2 \ldots$ where $\mu_{i+1} = \mu_i\Delta(a_i)$ and μ_0 is the initial distribution of $\mathcal{M}$. The set of all paths induced by $\mathcal{M}$, denoted by $Paths(\mathcal{M})$, is defined as the set $\{path(\sigma) \mid \sigma \in Act^\omega\}$. A distribution μ is said to be *i-reachable* if there is path of $\mathcal{M}$, $\mu_0\mu_1\mu_2 \ldots$ such that $\mu_i = \mu$. μ is said to be reachable if it is i-reachable for some i.

Labeling. A labeling for $\mathcal{M} = (Q, Act, \Delta, \mu_0)$ is a function $\lambda : Dist(Q) \to 2^{AP}$, where AP is a finite set of atomic propositions. A labeling function λ is binary iff $\forall p \in AP : U_p = \{\mu \in Dist(Q) \mid p \in \lambda(\mu)\}$ is a binary set.

The *trace* of a sequence of distributions $\pi = \mu_0\mu_1\mu_2 \ldots$ w.r.t λ is defined as the infinite sequence $\lambda(\pi) = \lambda(\mu_0)\lambda(\mu_1)\lambda(\mu_2) \ldots$. The set of all traces induced by $\mathcal{M}$ and λ, denoted by $Traces_\lambda(\mathcal{M})$, is defined as the set $\{\lambda(\pi) \mid \pi \in Paths(\mathcal{M})\}$.

Consider a binary labeling λ and the proposition $p \in AP$. Let U_p be parameterized by $a_1, a_2, \ldots, a_n, b$. The proposition p is said to be *robust* w.r.t λ if there exists an ϵ such that for every reachable distribution μ the following are equivalent

- $p \in \lambda(\mu)$ (i.e. $\sum_i a_i \mu(s_i) > b$)
- $\sum_i a_i \mu(s_i) > b + \epsilon$
- $\sum_i a_i \mu(s_i) > b - \epsilon$

More generally, the proposition is said to be *limit robust* w.r.t λ if there exists an ϵ and an m such that for every i-reachable distribution μ where $i \geq m$, the above three conditions are equivalent. The labeling λ is said to be (limit) robust, if every $p \in AP$ is (limit) robust w.r.t λ. For a (limit) robust λ, the minimum of the ϵ across all $p \in AP$ is known as the degree of robustness of λ.

We define the model checking problem for MDPs and labeling functions using Büchi automata as the specification.

Definition 7.1 (Model Checking Problem). *Given an MDP $\mathcal{M} = (Q, Act, \Delta, \mu_0)$, and a labeling function λ over AP, and a Büchi automaton $\mathcal{A}$ over the alphabet 2^{AP} determine if $Traces_\lambda(\mathcal{M}) \cap L(\mathcal{A}) = \emptyset$.*

In the above definition the Büchi automaton specifies the set of bad traces, and the question we are trying to answer is if the MDP and labeling produces a trace that is bad.

7.1.3 Complexity Classes

The complexity class RP consists of problems which can be solved using a randomized polynomial time algorithm that always returns "no" on no-instances, and returns "yes" with probability at least $\frac{1}{2}$ on yes-instances. We know that RP is contained in NP.

The counting hierarchy CH is a class of decision problems contained within PSPACE, which was introduced by Wagner [60]. The 0-th level, $\mathsf{C}_0\mathsf{P}$, is defined as P. The k-th level of the hierarchy is denoted by $\mathsf{C}_k\mathsf{P}$ and is defined recursively as $\mathsf{C}_{k+1}\mathsf{P} = \mathsf{PP}^{\mathsf{C}_k\mathsf{P}}$. Here PP denotes the class of decision problems for which there are polynomial time randomized

algorithms which answer "yes" with probability $> \frac{1}{2}$ on yes-instances, and answer "no" with probability $\geq \frac{1}{2}$ on no-instances. The whole counting hierarchy is contained in PSPACE.

In this paper we will assume every rational number is represented as $\frac{p}{q}$ where p and q are integers in binary. So, when we say a rational r can be computed in polynomial time given rationals $r_1, \ldots, r_k$, it implies that r can also be represented using polynomially many bits in the inputs $r_1, \ldots, r_k$.

7.1.4 Straight Line Programs

We will use *straight line programs* (SLP) to represent the computation of quantities such as acceptance probability of a word. A SLP over a set of variables V is a sequence of statements of the form $x := E$ where $x \in V$; E is either a constant in $\{0, 1\}$, a variable in V, or an expression of the form $e_1 \circ e_2$ where the operator $\circ \in \{+, -, *\}$ and $e_i \in \{0, 1\} \cup V$. Furthermore, each variable occurring on the right hand side of an assignment must occur in the left hand side of (some) earlier assignment. The value of a SLP is defined as the value assigned in its last statement. EquSLP is the problem of deciding if the value returned by the SLP is 0. PosSLP is defined as the problem of determining whether the value of the given SLP is positive. EquSLP was shown to be in coRP in [49]. A recent result [2] shows that PosSLP is in $\mathsf{P}^{\mathsf{C_3P}}$ and hence in the 4-th level of counting hierarchy.

7.2 Summary of Results for MDPs

In this Section we will briefly go through the results for problems pertaining to MDPs. These results follow from some well known results on probabilistic finite automata (PFA) which we shall touch upon. Most of our contribution in this thesis will be results regarding Markov chains, and we provide the results for MDPs here to show a contrast between the complexities.

7.2.1 Undecidability

The first result we observe is that the model checking problem for MDPs even when the labeling are restricted to be binary is undecidable. This result relies on the undecidability of the emptiness checking of probabilistic finite automata (PFA) by Condon and Lipton [17]. Therefore we review the definitions related to PFAs.

A PFA is a tuple $A = (Q, \Sigma, (\delta_\sigma)_{\sigma\in\Sigma}, \mu_0, F)$, where Q is a finite set of states, Σ is the finite input alphabet, $F \subseteq Q$ is the set of final states and $\mu_0 \in Dist(Q)$ is the initial distribution and $(\delta_\sigma)_{\sigma\in\Sigma}$ is an indexed set of stochastic matrices with dimension $|Q| \times |Q|$. For a symbol a, $\delta_a(s,t)$ represents the probability of going from state s to t on input symbol a. For any input word $w \in \Sigma^*$ of length n the probability of going from s to t along $w = a_1a_2\cdots a_n$ is then given by $\delta_w(s,t)$ where δ_w is the matrix $(\delta_{a_1} \cdot \delta_{a_2} \cdots \delta_{a_n})$. The distribution reached on input $w \in \Sigma^*$ in A is then given by $\mu_0\delta_w$.

The acceptance probability of a word $w \in \Sigma^*$ on PFA A is given by $\sum_{q\in F}(\mu_0\delta_w)(q)$ or $\mu_0\delta_w\eta_F$ where η_F is the column vector such that $\eta_F(j) = 1$ if $j \in F$ and $\eta_F(j) = 0$ otherwise. We will say that η_F is the vector corresponding to F.

Given a cut-point $t \in [0,1]$, the language of F w.r.t cut-point t is defined as the set $L_{>t}(A) = \{w \in \Sigma^* \mid \mu_0\delta_w\eta_F > t\}$. The result by Condon and Lipton [17] is that the emptiness problem of PFAs for any non-zero cut-point is undecidable.

Theorem 7.1 (Condon and Lipton [17]). *Given a PFA A the problem of checking if $L_{>\frac{1}{2}}(A) = \emptyset$ is undecidable.*

Using this we show the undecidability of the model checking problem:

Theorem 7.2. *The problem of model checking a MDP $\mathcal{M}$ with respect to a binary labeling function λ is undecidable.*

Proof. We reduce the emptiness problem for PFAs to our model checking problem. Let $\mathcal{F} = (Q, \Sigma, (\delta_\sigma)_{\sigma\in\Sigma}, \mu_0, F)$ be a PFA over alphabet Σ. We will construct a MDP $\mathcal{M} = (Q, Act, \Delta, \mu_0)$ which has the same state space as $\mathcal{F}$, the set of actions $Act = \Sigma$, the transition function $\Delta(q,a)$ is the q^{th} row of δ_a, and the initial distribution μ_0 is the same. Observe that

if there is a finite path of $\mathcal{M}$, $\mu_0\mu_1 \dots \mu_n$ obtained from the schedule $w = w_0 w_1 \dots w_n$ then, the word w is accepted with probability $\mu_n(F) = \mu_0 \delta_w \eta_F$ by $\mathcal{F}$, and vice versa.

Let $AP = \{p\}$ be the singleton proposition (label), and consider the labeling function λ such that $p \in \lambda(\mu)$ iff $\sum_{q \in F} \mu(q) > \frac{1}{2}$. Such a labeling is indeed binary and from the observation of the previous paragraph we obtain that $L_{>\frac{1}{2}}(\mathcal{F})$ is empty iff $\mathcal{M}$ has a path that reaches a distribution labeled p. □

7.2.2 Decidability under Robustness

Next, we observe that model checking MDPs becomes decidable when we restrict the binary labeling to be robust. We make use of the celebrated result by Rabin [48] which proves languages accepted by PFAs with isolated cut-points is regular. A cut-point t is said to be ϵ-isolated if there exists an $\epsilon > 0$ such that for all $w \in \Sigma^*$, $|\mu_0 \delta_w \eta_F - t| > \epsilon$.

Theorem 7.3 (Rabin [48]). *Given a PFA $\mathcal{F}$ with n states and r final states, and a cut-point t which is ϵ-isolated, the language $L_{>t}(\mathcal{F})$ is regular and is recognized by a DFA with $(1 + (r/\epsilon))^{(n-1)}$ states.*

Theorem 7.4. *The problem of model checking a MDP $\mathcal{M}$ with respect to a robust binary labeling λ where the degree of robustness ϵ is given is solvable in* EXPTIME.

Proof. Let a particular $p \in AP$ be parameterized by $a_1, \dots, a_n, b$ for the binary set it represents. Now we can define a PFA over the alphabet Act, where the states and transitions are given by the MDP $\mathcal{M}$, final states are those whose coefficients a_i are 1, and the cut-point for the PFA is b. Observe that a word $w \in Act^*$ is accepted by the PFA iff the distribution that the MDP reaches via w is such that p holds true on it according to λ. Now since λ is robust, we have that the PFA is isolated. Using Theorem 7.3 we can construct an exponential sized DFA that accpets the same language as the PFA. The DFA accepts a word $w \in Act^*$ iff $p \in \lambda(\mu_0 \delta_w)$. Therefore the synchronous product of these DFAs gives us a Moore machine $\mathcal{B}$ which outputs the labels according to λ along any infinite word in Act^ω. Taking a cross product of $\mathcal{B}$ with the Büchi automaton $\mathcal{A}$ of the specification where the output of the Moore machine $\mathcal{B}$ is fed into $\mathcal{A}$ gives us an exponential sized Büchi automaton $\mathcal{A} \times \mathcal{B}$. The automaton $\mathcal{A} \times \mathcal{B}$ over Act^ω accepts all and only all violating schedules. Model checking

then reduces to checking emptiness of which can be done time linear in the size of the graph. This gives us the EXPTIME upper bound. □

7.2.3 Undecidability of Checking Robustness

The problem of checking if a binary label is robust is equivalent to the problem of checking isolation of PFAs. Given MDP $\mathcal{M} = (Q, Act, \Delta, \mu_0)$ and a proposition p in a binary label λ where U_p is parameterized by $a_1, a_2, \ldots, a_n, b$, we construct a PFA $A = (Q, \Sigma, (\delta_\sigma)_{\sigma\in\Sigma}, \mu_0, F)$ as before where Q and μ_0 are the same, $\Sigma = Act$, δ_σ is the stochastic matrix whose q^{th} row is $\Delta(q, \sigma)$, and $F = \{q \in Q \mid a_q = 1\}$. A along with cut-point b is then isolated iff the proposition p is robust for $\mathcal{M}$. Simlarly given a PFA A and cut-point b one can construct a MDP and a labeling such that the two problems are identical. The isolation problem turns out to be undecidable, which gives us the undecidability of the robustness problem.

Theorem 7.5 ([9, 25, 11]). *Given PFA $\mathcal{F}$ and cut-point t, the problem of deciding whether t is isolated for $\mathcal{F}$ is Σ_2^0-complete (undecidable).*

Theorem 7.6. *The problem of checking whether a given proposition is robust w.r.t binary label λ and MDP $\mathcal{M}$ is Σ_2^0-complete.*

Corollary 7.1. *The problem of checking whether a given proposition is limit robust w.r.t binary label λ and MDP $\mathcal{M}$ is undecidable.*

7.2.4 Roadmap

In the remaining chapters we consider the model checking and the robustness problems for Markov Chains. First we prove that the problem of checking limit robustness of a given binary label for Markov Chains is coNP-complete. We also prove that the robustness checking problem for a binary label for Markov Chains is $\mathsf{coNP}^{\mathsf{RP}}$. Both these results are covered in Chapter 8. Regarding the model checking problem, we prove that it is decidable in PSPACE for Markov chains against binary labels that are limit robust in Chapter 9.

Chapter 8

Complexity of Checking Robustness

In this chapter we look at the question of deciding whether a proposition in a binary labeling is robust or limit robust for a given Markov chain. Recall that the truth of proposition in a binary labeling is derived from an inequality of the form $\sum_i a_i \mu(s_i) > \theta$ where μ is a probability distribution over the states $s_0, \ldots, s_n$ and each $a_i \in \{0, 1\}$ and $\theta \in [0, 1]$ are constants. We will borrow some vocabulary from the study of PFAs to make our statements simpler. The set of states $F = \{s_i \mid a_i = 1\}$, will be referred to as final states for the given proposition. We shall refer to the left hand side of the inequality $\sum_i a_i \mu(s_i)$ as the acceptance probability for a given distribution, and we shall refer to the right hand side θ as the cut-point. The acceptance probability after n steps is defined as the acceptance probability of the distribution reached after n steps in the Markov chain. A proposition is said to be robust if there is some ϵ such that changing the right hand side of the inequality to $\theta \pm \epsilon$ does not change the truth of the proposition on any of the reachable distributions. In other words a proposition is robust if the acceptance probability is always bounded away from θ. A proposition is said to be limit robust if the acceptance probability is bounded away from θ for all but finitely many reachable distributions.

We briefly summarize our technique before we delve into the specifics in the rest of the chapter. When $\theta \in (0, 1)$, we show that the robustness problem is decidable and is in $\mathsf{coNP}^{\mathsf{RP}}$. Note that since RP is contained in NP (see [5]), this implies that the problem of checking robustness when $\theta \in (0, 1)$ is in the second level of polynomial hierarchy. Furthermore, given that RP is believed to be P, this would imply the problem to be in coNP which matches the lower bound of coNP-hardness mentioned ahead. Our procedure also gives a way to compute a degree of robustness if the proposition is robust for the Markov chain. Our result is proved as follows. If a proposition is limit robust then there is a $n_0 > 0$ such that the acceptance

probability after n steps is bounded away from θ for all $n > n_0$. Thus the proposition is robust iff it is limit robust and the probability of reaching the final states in a "short" time (i.e., in time less than n_0) is bounded away from θ. We first prove (in Section 8.1) that the problem of checking if a proposition is limit robust is in coNP . Next, we show that if it is limit robust, then the bound n_0 is "small" (Section 8.2). More precisely, we show that this number n_0 can be represented in binary using polynomially many bits (in the size of Markov chain). Using this observation, we can conclude that if a proposition is limit robust for a Markov chain, then it is not robust iff there is some number (which at most is exponentially large) such that acceptance probability in those many steps is exactly θ. The check of whether the acceptance probability after ℓ is equal to θ can be reduced to checking if a straight-line program (Section 7.1.4) of length ℓ using addition, multiplication, and subtraction computes a real number that is equal to 0. Based on this observation, and results on the complexity of the EquSLP problem [49], we conclude that checking robustness for a proposition in a binary labeling of a Markov chain is in $\mathsf{coNP}^{\mathsf{RP}}$.

8.1 Checking Limit Robustness

First, we prove that the problem of checking limit robustness of a proposition in a binary labeling for Markov Chains is coNP-complete. In order to prove these results, we recall some standard facts about Markov chains. The proofs of these facts can be found in [23].

Theorem 8.1. *Let $c \in \mathcal{C}$ be a BSCC of a Markov Chain $M = (Q, \delta, \mu_0)$, p be the period of c, then for any state j in c:*

1. *If i is a transient state of M then $\lim_{r\to\infty} \delta^{pr}(i, j)$ exists and can be calculated in time polynomial in the size of δ.*

2. *If i is in c, then $\lim_{r\to\infty} \delta^{pr}(i, j)$ exists and can be calculated in time polynomial in the size of δ.*

3. *If i is neither a transient state of M nor in c then $\lim_{r\to\infty} \delta^{pr}(i, j) = 0$ (in fact $\delta^{\ell}(i, j) = 0$ for all ℓ).*

This leads to the following corollary (recall that the ultimate period of δ is the l.c.m of the period of its BSCCs).

Corollary 8.1. *For any stochastic matrix δ, with ultimate period p, $\widehat{\delta^p} = \lim_{r\to\infty} \delta^{pr}$ exists.*

We are ready to show that limit robustness is coNP-complete.

Theorem 8.2. *The problem of checking given a Markov chain M and a proposition parameterized by $a_0, \ldots, a_n, \theta$, whether the proposition is limit robust is* coNP*-complete.*

Proof. (**Upper Bound**). Let $M = (Q, \delta, \mu_0)$ and let p be the ultimate period of δ. According to Corollary 8.1, there are possibly p different limits towards which the Markov chain approaches in a cyclic manner. That is for each $0 \leq k < p$, we have that $\lim_{r\to\infty} \delta^{k+pr}$ exists.

If the proposition is not limit robust then it is easy to see that there is a $0 \leq k < p$ such that $\lim_{r\to\infty} \mu_0\delta^{k+pr}\eta_F$ is θ. The witness for a *no* answer to our problem is therefore going to be this number k which requires only $n \log n$ bits to be represented.

The result will follow if we can compute the distribution $\mu_k\widehat{\delta^p} = \lim_{r\to\infty} \mu_0\delta^{k+pr}$ in polynomial time. This can be achieved as follows. Note that $\mu_k\widehat{\delta^p}(i) = 0$ for any transient state i. We only have to compute $\mu_k\widehat{\delta^p}(i)$ for terminal states i.

Consider a BSCC c_i of δ. Let its period be p_i, let k_i be $k \bmod p_i$. Note that p can be exponentially large but each of the p_is at most n. Although $\sigma_{p_i} = \delta^{p_i}, \delta^{2p_i}, \ldots$ need not converge, it follows from Theorem 8.1 that the columns corresponding to c_i do converge to a limit. Now $\widehat{\delta^{p_i}}(;,j) = \widehat{\delta^p}(;,j)$ for any state $j \in c_i$ because σ_p is a subsequence of σ_{p_i}. So the entire matrix $\widehat{\delta^p}$ can be calculated in polynomial time. Essentially the jth column of $\widehat{\delta^p}$ is identical to the jth column of $\widehat{\delta^{p_i}}$. In order to calculate $\delta^k\widehat{\delta^p}$ observe that its jth column $\delta^k\widehat{\delta^p}(;,j) = \delta^k\widehat{\delta^{p_i}}(;,j) = \delta^{k_i}\widehat{\delta^{p_i}}(;,j)$ where again p_i is the period of the BSCC c_i that contains j. Note that Now $k_i < p_i \leq n$ and so δ^{k_i} can be calculated in polynomial time. The upper bound follows.

(**Lower Bound**). In order to prove hardness we use the reduction in [27, 55] which is used to show coNP-hardness of the universality problem for unary non-deterministic finite automata (NFA). We briefly describe the salient features of the reduction; for further details the reader should refer to [27]. The original reduction is from 3SAT to non-universality of unary NFA. Given a 3SAT formula ϕ with n variables and m clauses, [27] constructs a

NFA N_ϕ as a union of m cyclic automata. Intuitively, each cycle corresponds to a clause, has an initial state and a cycle accepts if and only if the input encodes an assignment that does not satisfy that clause. So N_ϕ accepts every input iff ϕ is unsatisfiable. The only non-determinism in N_ϕ is from having to choose a cycle at the beginning, so we can transform it into a Markov chain M_ϕ by choosing amongst the cycles uniformly at random and define a proposition parameterized by final states of N_ϕ and $\theta = 0$. Since there are only m cycles if any word a^ℓ is accepted by N_ϕ then the acceptance probability after ℓ steps in M_ϕ will have acceptance probability at least $\frac{1}{m}$; otherwise it the acceptance probability after ℓ steps is 0. Therefore the proposition is robust for M_ϕ iff ϕ is unsatisfiable. Finally we observe that the proposition is robust iff it is limit robust, which proves that the proposition is limit robust for M_ϕ iff ϕ is unsatisfiable. We have already observed that if a proposition is robust then it is also limit robust. For the converse observe that the constructed unary NFA N_ϕ is a disjoint union of cycles. Let d be the lcm of all the cycles of N_ϕ. Now it is easy to see that for each j, the probability distribution on the states of the unary PFA P_ϕ after j steps is the same as the probability distribution on after $j \bmod d$ steps. So if there is some j-reachable distribution whose acceptance probability is 0, then there are infinitely many reachable distribution with 0 acceptance probability and therefore the considered proposition cannot be limit robust. □

Please note that the lower bound proof of Theorem 8.2 can be modified if the cut-point θ is not extremal; simply add an additional state with a self loop, which you choose initially with probability θ. Also, we could have taken the cut-point to be 1 by switching the final and non-final states. Thus, complexity of limit robustness does not depend on whether the cut-point is extremal or not. Also, note that the lower bound proof also establishes the coNP-hardness of the robustness problem.

8.2 Checking Robustness

We will prove that the problem of checking whether a proposition (parameterized by $a_1, \ldots, a_n$ and θ) is robust is in $\mathsf{coNP}^{\mathsf{RP}}$ (see Theorem 8.3). For extremal cut-points, i.e., when θ is 0 or 1, we will show the problem to be coNP-complete (see Theorem 8.4). We start by discussing non-extremal cut-points.

Non-extremal cut-points.

Broadly speaking, the proof for showing that checking robustness of a proposition is in $\mathsf{coNP}^{\mathsf{RP}}$ is as follows:

- We can use Theorem 8.2 to check if the proposition is limit robust. If it is not limit robust then we know that it is not robust.
- If it is limit robust, then robustness follows iff there is no number ℓ such that the acceptance probability after ℓ steps is exactly θ. We will show that that this number cannot be too long (see Lemma 8.2).
- We can then guess this number, construct a straight-line program such that its value is 0 iff the acceptance probability after ℓ steps is θ, and check if it evaluates to 0 or not (see Lemma 8.1).

We start by showing that the problem of deciding given a Markov chain M and a number n in binary, whether the acceptance probability, for a given set of final states, $=\theta$ is in coRP and $> \theta$ is in the counting hierarchy.

Lemma 8.1. *Given a Markov chain M and a proposition parameterized by $(a_1, \ldots, a_n, \theta)$, and a non-negative integer n in binary as input, the problem of checking:*

1. *if the acceptance probability after n steps is equal to θ is in* coRP.
2. *if acceptance probability after n steps is greater than θ lies in* $\mathsf{P}^{\mathsf{C}_3\mathsf{P}}$.

Proof. The acceptance probability after n steps is $\mu_0 \delta^n \eta_F$, where μ_0 is the initial distribution, δ the transition matrix and η_F the vector corresponding to the final states (states i where $a_i = 1$). In order to find out if this quantity is equal to θ, one can write a straight line program p that calculates $\mu_0 \delta^n \eta_F - \theta$. The program is the usual square-and-multiply algorithm for exponentiation and it is going to be $O(\log_2 n)$ long because the number of iterations in the algorithm is equal to the number of bits required to represent n. The value of the program p is equal to (greater than) 0 iff acceptance probability after n steps is exactly equal to (greater than) θ. Now, we can check if $val(p) = 0$ in coRP [49] and $val(p) > 0$ in $\mathsf{P}^{\mathsf{C}_3\mathsf{P}}$ [2]. The result follows. □

We will now show that if a limit robust proposition is such that the acceptance probability after n steps is exactly θ then n cannot be too large. This fact is proved in Lemma 8.2 with the help of auxiliary Propositions 8.1, 8.2 and 8.3. We start by proving a result about irreducible stochastic matrices. Recall that $\widehat{\delta^t}$ is used to denote the limit of the sequence $\lim_{r\to\infty} \delta^{rt}$.

Proposition 8.1. *Given an irreducible stochastic matrix δ with period p and rational $\epsilon \in (0,1)$ there exists a number k, computable in polynomial time, such that for all $\ell \geq k$: $\boldsymbol{d}(\delta^{p\ell}, \widehat{\delta^p}) \leq \epsilon$.*

Proof. A stochastic matrix γ with all positive entries acts as a contraction map on the set of distributions. The associated contraction factor α is $(1-ns)$ where s is the smallest entry in γ (see Proposition 7.2). So we have

$$\begin{aligned}\boldsymbol{d}(\mu\gamma^i, \mu\widehat{\gamma}) = \lim_{j\to\infty} \boldsymbol{d}(\mu\gamma^i, \mu\gamma^j) &\leq \lim_{j\to\infty} \sum_{i'=i}^{j-1} \boldsymbol{d}(\mu\gamma^{i'}, \mu\gamma^{i'+1}) \\ &\leq \lim_{j\to\infty} \sum_{i'=i}^{j-1} \alpha^{i'}\, \boldsymbol{d}(\mu, \mu\gamma) \leq \frac{\alpha^i}{1-\alpha} = \frac{(1-ns)^i}{ns} \leq \frac{e^{-nsi}}{ns}.\end{aligned}$$

Choosing $i > \frac{1}{ns}\log\frac{2}{ns\epsilon}$ will give us $\boldsymbol{d}(\mu\gamma^i, \mu\widehat{\gamma}) \leq \frac{\epsilon}{2}$ and because the μ is arbitrary we also have $\boldsymbol{d}(\gamma^i, \widehat{\gamma}) \leq \epsilon$.

Coming back to δ, the graph of δ^p consists of p disjoint irreducible and aperiodic components. It is enough to show the above bound on each of the individual components (because the distance between the matrices takes maximum across rows), so consider δ^p to be irreducible and aperiodic. From Lemma 7.1, we know that δ^{pn^2} has all positive entries. The smallest entry of δ^{pn^2}, say s, requires only polynomially many bits to be represented. According to the above observation, for $i \geq \frac{1}{ns}\log\frac{2}{ns\epsilon}$ we have $\boldsymbol{d}(\delta^{pn^2 i}, \widehat{\delta^p}) \leq \epsilon$. If $\frac{1}{ns\epsilon} = \frac{x}{y}$, and j represents the number of bits of y then we can choose $k = \lceil \frac{n}{s}(j+1) \rceil$, which is computable in polynomial time. □

We now bound the number of steps required so that the probability of being in a transient state is small.

Proposition 8.2. *Given a stochastic matrix δ and rational $\epsilon \in (0,1)$ there exists a number k, computable in polynomial time such that for all $\ell \geq k$ it is the case that for all distributions μ_0, $\sum_{j \in T_\delta} \mu_0 \delta^\ell(j) \leq \epsilon$ where T_δ is the set of transient states of δ.*

Proof. Here we are required to show that after k steps the probability of being in a transient state is small. Every transient state has a path of length at most n to at least one terminal state, so choose one for each transient state. Let u be the minimum probability associated with any of those paths. So after every n steps each transient state loses at least u fraction of its probability to a terminal state, or in other words the probability of being in any transient state reduces by a factor of u. Hence after $k'n$ steps the probability of being in a transient state is at most $(1-u)^{k'}$, and choosing $k' \geq \frac{1}{u} \log \frac{1}{\epsilon}$ makes $(1-u)^{k'} \leq \epsilon$. So choosing k to be a number bigger then $\frac{n}{u} \log \frac{1}{\epsilon}$ we have our required number. □

We now bound the length of input needed to be close to the limit distribution $\mu\widehat{\delta^p}$ where p is the ultimate period of δ.

Proposition 8.3. *Given a stochastic matrix δ, a distribution μ and rational $\epsilon \in (0,1)$ there exists a k, computable in polynomial time such that for all $\ell \geq k$: $\boldsymbol{d}(\mu\delta^{p\ell}, \mu\widehat{\delta^p}) \leq \epsilon$ where p is the ultimate period of δ.*

Proof. (Sketch.) First we use Proposition 8.2 to get a k_1 such that it suffices to take k_1 steps to get to a distribution where the probability of being in any transient state is less than $\frac{\epsilon}{4}$. This ensures that for $l \geq k_1$, the probability of being in any BSCC c after pl steps is at least $1 - \frac{\epsilon}{4}$. This means that taking any more steps beyond k_1 can only perturb the probability in terminal states by a small amount which adds up to $\frac{\epsilon}{4}$ across all BSCCs. Let us focus on one BSCC c. Taking, k_2 steps beyond the k_1 will do two things to c:

i) bring in more probability from the transient states

ii) distribute the probability already present in c (i.e., the probability of being in c) at step k_1 according to μ_c, the stationary distribution of c.

The first effect can only result in pumping at most a small probability into c, which adds at most $\frac{\epsilon}{4}$ to the distance. The probability already present in c after k_1 steps is close to the

limiting probability, and hence the contribution of the second effect into the distance can be made small by choosing k_2 according to Proposition 8.1 for the BSCC c with the bound $\frac{\epsilon}{4}$. Instead of choosing k_2 for a particular c, we can choose it to be the maximum across all c which will give us the desired result. We formalize these ideas in the calculations below.

For the purposes of the calculation below let a *sub-distribution* over a finite set S is a function $\mu : S \mapsto [0,1]$ such that $\sum_{s\in S} \mu(s) \leq 1$. The distance between sub-distributions can be defined in the same way we do for distributions.

We first describe the notation we will use in the following calculations: $\mu_{\mathcal{T}}$ and $\mu_{\mathcal{C}}$ denote the sub-distributions on the transient states and the BSCCs after pk_1 steps. For any (sub-)distribution μ , $\mu \restriction c$ denotes the vector with the entries in the states in c alone. The matrix δ restricted to the states of c is written as δ_c. Starting from μ, and having taken pk_1 steps, the probability of being in component c is denoted by $\pi_c^{pk_1}$, and the relative distribution on a component c is given by $\mu_c^{pk_1}$, i.e for any $i \in c$, $\mu_c^{pk_1}(i) = \mu\delta^{pk_1}(i)/\pi_c^{pk_1}$. Starting from μ the probability of being in c in the limit is given by $\hat{\pi_c}$ and the relative distribution on c in the limit is give by $\hat{\mu_c}$. Now we are ready to proceed.

$$\boldsymbol{d}\big(\mu\delta^{p\ell}, \mu\widehat{\delta^p}\big) = \boldsymbol{d}\big(\mu\delta^{pk_1}\delta^{pk_2}, \mu\widehat{\delta^p}\big) = \boldsymbol{d}\big((\mu_{\mathcal{T}} + \mu_{\mathcal{C}})\delta^{pk_2}, \mu\widehat{\delta^p}\big)$$

$$= \boldsymbol{d}\big(\mu_{\mathcal{T}}\delta^{pk_2} + \mu_{\mathcal{C}}\delta^{pk_2}, \mu\widehat{\delta^p}\big) \leq \underbrace{\frac{\sum_j \mu_{\mathcal{T}}\delta^{pk_2}(j)}{2}}_{\text{Apply Prop 8.2}} + \boldsymbol{d}\big(\mu_{\mathcal{C}}\delta^{pk_2}, \mu\widehat{\delta^p}\big)$$

Let us focus on $\boldsymbol{d}\big(\mu_{\mathcal{C}}\delta^{pk_2}, \mu\widehat{\delta^p}\big)$

$$= \sum_{c\in\mathcal{C}} \boldsymbol{d}\big((\mu_{\mathcal{C}}\delta^{pk_2}) \restriction c\,, (\mu\widehat{\delta^p}) \restriction c\big) = \sum_{c\in\mathcal{C}} \boldsymbol{d}\big(\pi_c^{pk_1}\mu_c^{pk_1}\delta_c^{pk_2},\ \hat{\pi}_c\,\hat{\mu_c}\widehat{\delta_c^p}\big)$$

We have $(\mu_{\mathcal{C}}\delta^{pk_2}) \restriction c \;= \pi_c^{pk_1}\mu_c^{pk_1}\delta_c^{pk_2}$ because when we start from $\mu_{\mathcal{C}}$ there is no probability of being in any transient state, so we can ignore the transient states, and then the BSCCs cannot communicate so they evolve independently.

$$
\begin{aligned}
&= \sum_{c\in\mathcal{C}}\sum_{j\in c}\left|\pi_c^{pk_1}\mu_c^{pk_1}\delta_c^{pk_2}(j) - \hat{\pi}_c\,\hat{\mu}_c\widehat{\delta_c^{p}}(j)\right| \\
&= \sum_{c\in\mathcal{C}}\sum_{j\in c}\left|\pi_c^{pk_1}\left(\mu_c^{pk_1}\delta_c^{pk_2}(j) - \hat{\mu}_c\widehat{\delta_c^{p}}(j)\right) + (\pi_c^{pk_1} - \hat{\pi}_c\,)\hat{\mu}_c\widehat{\delta_c^{p}}(j)\right| \\
&\leq \left(\sum_{c\in\mathcal{C}}\pi_c^{pk_1}\sum_{j\in c}\left|\mu_c^{pk_1}\delta_c^{pk_2}(j) - \hat{\mu}_c\widehat{\delta_c^{p}}(j)\right|\right) + \left(\sum_{c\in\mathcal{C}}(\hat{\pi}_c - \pi_c^{pk_1})\sum_{j\in c}\hat{\mu}_c\widehat{\delta_c^{p}}(j)\right) \\
&\leq \left(\sum_{c\in\mathcal{C}}\pi_c^{pk_1}\overbrace{\boldsymbol{d}\big(\mu_c^{pk_1}\delta_c^{pk_2}(j),\,\hat{\mu}_c\widehat{\delta_c^{p}}(j)\big)}^{\text{Apply Prop 8.1}}\right) + \overbrace{\sum_{c\in\mathcal{C}}(\hat{\pi}_c - \pi_c^{pk_1})}^{\text{Apply Prop 8.2}} \\
&\leq \left(\sum_{c\in\mathcal{C}}\pi_c^{pk_1}\frac{\epsilon}{2}\right) + \frac{\epsilon}{4} \leq \frac{3\epsilon}{4}
\end{aligned}
$$

Therefore $\boldsymbol{d}\big(\mu\delta^{\ell}, \mu\widehat{\delta^{p}}\big) \leq \epsilon$. □

We can prove a similar result about the matrix products as well.

Lemma 8.2. *Given a stochastic matrix δ and a rational $\epsilon \in (0,1)$, there exists a number k, computable in polynomial time, such that for all $\ell \geq k : \boldsymbol{d}(\delta^{p\ell}, \widehat{\delta^{p}}) \leq \epsilon$ where p is the ultimate period of δ.*

Proof. The distance between the matrices can be broken into

$$
\begin{aligned}
\boldsymbol{d}(\delta^{p\ell}, \widehat{\delta^{p}}) &= \max_i \sum_j |\delta^{p\ell}(i,j) - \widehat{\delta^{p}}(i,j)| = \max_i \sum_j |\nu_i\delta^{p\ell}(j) - \nu_i\widehat{\delta^{p}}(j)| \\
&= \max_i \left(2\,\boldsymbol{d}(\nu_i\delta^{p\ell}, \nu_i\widehat{\delta^{p}})\right). \qquad \text{Here } \nu_i \text{ represents the distribution with probability 1 at state } i
\end{aligned}
$$

Proposition 8.3 tells us we can choose a k of appropriate size such that for any μ, the distance $\boldsymbol{d}(\mu\delta^{p\ell}, \mu\widehat{\delta^{p}})$ for $\ell \geq k$ is below $\frac{\epsilon}{2}$. □

We are ready to establish the complexity of the problem of checking if a proposition in a binary labeling for a Markov chain is limit robust.

Theorem 8.3. *Given a Markov chain M and a proposition in a binary labeling parameterized by $a_1, \ldots, a_n, \theta$, the problem of checking if the proposition is robust is in* $\mathsf{coNP}^{\mathsf{RP}}$ *and is* coNP*-hard.*

Proof. The lower bound follows from the proof of Theorem 8.2. For the $\mathsf{coNP}^{\mathsf{RP}}$ upper bound, let us consider the complement of the problem where the proposition is not limit robust. In this case either the proposition is not limit robust or there is some n such that is the acceptance probability (as per $a_1, \ldots, a_n$) after n steps is exactly θ. If the proposition is limit robust then we guess this fact and check if it is true in NP (Thanks to Theorem 8.2). So assume that it is limit robust. We now need to check if there is any "small" number of steps such that probability of acceptance after those many steps is θ.

Let $M = (Q, \delta, \mu_0)$ and let p be the ultimate period of δ. Let $\widehat{\delta^p} = \lim_{t\to\infty} \delta^{pt}$. For each $r > 0$, let $\mu_r = \mu_0 \delta^r$.

Consider $\epsilon_r = |\mu_r \widehat{\delta^p} \eta - \theta|$ where η is the final states or the vector corresponding to $a_1, \ldots, a_n$. Since any $\mu_r \widehat{\delta^p}$ can be computed in polynomial time (see proof of Theorem 8.2), it is the case that ϵ_r can be computed in polynomial time (given μ_0, δ, r). Suppose the length of the string accepted with probability θ is ℓ. Let $\ell = pq + r$ where $r = \ell \bmod p$. According to Lemma 8.2, there exists a k_r (computable in polynomial time) such that if $q > k_r$ then $\boldsymbol{d}(\mu_r \delta^{pq}, \mu_r \widehat{\delta^p}) \leq \frac{\epsilon_r}{2}$. Since $\boldsymbol{d}(\mu_r \delta^{pq}, \mu_r \widehat{\delta^p}) \geq |\mu_r \delta^{pq} \eta - \mu_r \widehat{\delta^p} \eta|$, we get that acceptance probability after ℓ steps is θ if $q > k_r$.

Now, the decision procedure for checking non-robustness proceeds as follows. It first guesses $0 \leq r < p$, then it computes ϵ_r and subsequently computes k_r. Now, it guesses $q \leq k_r$ and then it computes $\ell = pq + r$. ℓ requires only polynomially many bits (because k_r is computable in polynomial time from r). Hence we can use the procedure of Lemma 8.1 as an oracle to check if the acceptance probability after ℓ steps is exactly θ. Note that this final check is done by a coRP algorithm and hence the non-robustness is in $\mathsf{NP}^{\mathsf{coRP}}$. Note that $\mathsf{NP}^{\mathsf{coRP}}$ is exactly the class $\mathsf{NP}^{\mathsf{RP}}$ since we can always switch the yes/no answer from the oracle-calls. This results in a $\mathsf{coNP}^{\mathsf{RP}}$ upper bound for the limit robustness problem in the non-extremal case. □

Extremal cut-points.

For extremal cut-points, the upper bound matches the lower bound.

Theorem 8.4. *Given a Markov chain M, the problem of checking if a proposition, parameterized by $a_1, \ldots, a_n, \theta$ and $\theta = 0$, is robust is* coNP*-complete. Similarly checking robustness when $\theta = 1$ is robust is also* coNP*-complete.*

Proof. The lower bound follows from the proof of Theorem 8.2. For upper bound, first thing to note is that the coNP upper bound for limit robustness proved in Theorem 8.2 also holds for the $\theta = 0$. So in case it is limit robust, we need to check if there is a number ℓ such that acceptance probability after ℓ steps is 0. If μ_0 is the initial distribution, let $Q_I = \{q \mid \mu_0(q) > 0\}$. The probability of acceptance after ℓ steps is 0 iff there is no path from a state in Q_I to a final state in exactly ℓ steps. So checking robustness when $\theta = 0$ reduces to the universality checking of unary NFA which is known to be in coNP [55]. □

Chapter 9

Decidability under Limit Robustness

In this chapter we will discuss results for the model checking problem for Markov chains and the containment problem unary PFAs under the assumption of limit robustness for Markov chains and isolation for PFAs. Our main proposition is as follows:

Proposition 9.1. *Given a Markov Chain $\mathcal{M}$ with ultimate period p, a number $0 \leq r < p$ and a limit robust proposition in a binary labeling, parameterized by $a_1, a_2, \ldots, a_n, b$ there is a number k computable in polynomial time s.t $\forall q \geq k$, $\mu\delta^{pq+r}\vec{a} > b$ iff $\mu_0\delta^{pk+r}\vec{a} > b$, where $\vec{a}$ is the column vector $(a_1, a_2, \ldots, a_n)$.*

Proof. Let $\mathcal{M} = (Q, \delta, \mu_0)$. Let $\mu_r = \mu_0\delta^r$, let $\widehat{\delta^p} = \lim_{t\to\infty} \delta^{pt}$. Consider $\epsilon_r = |\mu_r\widehat{\delta^p}\vec{a} - b|$. ϵ_r can be computed in polynomial time (see proof of Theorem 8.2). According to Lemma 8.2, there is a k computable in polynomial time such that if $q > k$ then $\boldsymbol{d}(\mu_r\delta^{pq}, \mu_r\widehat{\delta^p}) \leq \frac{\epsilon_r}{2}$. Since $\boldsymbol{d}(\mu_r\delta^{pq}, \mu_r\widehat{\delta^p}) \geq |\mu_r\delta^{pq}\vec{a} - \mu_r\widehat{\delta^p}\vec{a}|$, we get that a^{pq+r} has acceptance probability $> b$ iff a^{pk+r} has acceptance probability $> b$. □

A similar result holds for unary PFAs with limit robust cut-points.

Corollary 9.1. *Given a PFA A with ultimate period p, a number $0 \leq r < p$ and a rational cut-point θ such that θ is limit isolated for A, there is a number k computable in polynomial time s.t $\forall q \geq k$, $a^{pq+r} \in L_{>\theta}(A)$ iff $a^{pk+r} \in L_{>\theta}(A)$.*

Next we look at the model checking problem under the limit robustness restriction.

Theorem 9.1. *The model checking problem for Markov chains when the labeling is binary and restricted to be limit robust, is solvable in* PSPACE. *(the result holds even if the degree of limit robustness is not given)*

Proof. Let $\mathcal{M} = (Q, \delta, \mu_0)$ be the Markov chain and λ be the limit robust binary labeling. Let p be the ultimate period of δ. Note that by Proposition 9.1 there is a k of polynomially many bits such that $\forall r : 0 \leq r < p$ and $\forall q \geq k$, $\lambda(\mu_0\delta^{pq+r}) = \lambda(\mu_0\delta^{pk+r})$, and k can be computed in PSPACE. Since $\mathcal{M}$ is a Markov chain, it induces a single path $\mu_0\mu_1\mu_2\ldots$, and hence yields a single trace $\lambda(\mu_0)\lambda(\mu_1)\lambda(\mu_2)$. From our last observation this trace is of the form $\tau = uv^\omega$, where $|u| = k$ and $|v| = p$. The model checking problem now boils down to checking whether there is an accepting run for the word τ over the specification given as a Büchi automaton $\mathcal{A}$. For any accepting run α (for τ) let Inf-Step(α) denote the set $\{q \in Q_\mathcal{A} \mid q = \alpha_{|u|+i|v|}$ for infinitely many $i\}$. Consider all the those runs α which have the earliest (across all α) occurence of some state in Inf-Step(α) at some position $|u|+i|v|$, denote this set by $\mathcal{E}$ and the earliest index as e. Note that this earliest position $e \leq |u| + |Q_\mathcal{A}||v|$, otherwise there is going to be a repeated state at $|u| + i_1|v|$ and $|u| + i_2|v|$ $(i_1 < i_2 \leq |Q_\mathcal{A}|)$ due to pigeon hole principle and we can "short circuit" the path to get a run α' over τ in which an earlier position has an occurence of Inf-Step(α'). Now, let Inf-Final(α) denote the set $\{q \in F_\mathcal{A} \mid q = \alpha_i$ for infinitely many $i\}$. Consider all those runs β in $\mathcal{E}$ where the first occurence after e of some state in Inf-Final(β) is the earliest across all β. Now using a similar argument as above we can say that this earliest position f is such that $f \leq e + (|Q_\mathcal{A}| + 1)|v|$. Now among all such β we look at those runs in which the first occurence after f, of the state at position e in β, is the earliest across all runs β. Let this position be g, again we can argue that $g \leq f + (|Q_\mathcal{A}| + 1)|v|$. Thus we obtain a finite run over some uv^j of length g, such that the state at positions g and e (which corresponds to uv^i for some $i < j$) are the same, and there is some final state at position f such that $e < f < g$. Now one can extend this finite run to uv^ω by repeating the portion between e and g to obtain an accepting run. Thus the string uv^j along with positions e, f and g is a finite witness of an infinite accepting run. Note that the positions can be represented in polynomially many bits but the string uv^j, which is a prefix of τ, is exponentially long. But we observe that the i^{th} element in the trace τ_i can be computed from i alone in PSPACE, thanks to part 2 of Lemma 8.1. Thus we can guess e, f and g at the beginning, and guess the prefix of the accepting run over τ, transition by transition, and check that (i) each transition is true to $\mathcal{A}$, (ii) $|u| \leq e < f < g$, (iii) $e \equiv g \bmod |v|$, (iv) states at e and g match and (v) state at f is final, all in PSPACE. □

Next we present results pertaining to decision problems for Unary PFAs.

Theorem 9.2. *Given two unary PFAs A and B (over the same alphabet) and rational cut-points θ_1 and θ_2, such that θ_1 and θ_2 are limit isolated for A and B respectively, the following problems are in* $\mathsf{coNP}^{\mathsf{C_3P}}$.

1. $L_{>\theta_1}(A) \subseteq L_{>\theta_2}(B)$.
2. $L_{>\theta_1}(A) = L_{>\theta_2}(B)$.
3. $L_{>\theta_1}(A) = \emptyset$.
4. $L_{>\theta_1}(A) = \Sigma^*$.

Proof. Without loss of generality, we can assume that the ultimate periods of A and B are the same (since we can always add unreachable cycles). Let the ultimate period be p. The algorithm for checking containment proceeds as follows.The algorithm is going to guess a number ℓ such that a^ℓ (here a is the lone symbol in the unary alphabet) is accepted by A and rejected by B. Note, ℓ can be written as $\ell = pq + r$ where $q = \ell \ \mathsf{div}\ p$ and $r = \ell \ \mathsf{mod}\ p$. Hence we have to guess q and r. First, the algorithm guesses the offset $0 \leq r < p$ which is a polynomial-sized number.

Thanks to Corollary 9.1, there is a k_A such for all $q_A \geq k_A$, a^{pq_A+r} is accepted by A iff the string a^{pk_A+r} is accepted. Furthermore, k_A can be computed in polynomial time from A and r. Similarly, there is a k_B such that for all $q_B \geq k_B$, a^{pq_B+r} is accepted by B iff the string a^{pk_B+r} is accepted. Let $k = \max(k_A, k_B)$.

By construction, we can conclude that if a^ℓ with $\ell = pq+r$ is in the language of A but not in the language of B then we can take $q \leq k$. So, now the algorithm guesses $q \leq k$ and then checks that i) $a^\ell \in L_{>\lambda_1}(A)$ and ii) $a^\ell \notin L_{>\lambda_2}(B)$. These checks can be carried out by $\mathsf{P}^{\mathsf{C_3P}}$ algorithms as in Lemma 8.1 and the result follows. The other problems of language equality, emptiness, and universality follow immediately from the result for language containment. □

9.1 Conclusion

We conclude with a summary of results we have covered in this part of the thesis. In Table 9.1 we contrast the complexities of the model checking problem for MDPs against Markov chains under different restrictions. Each line of the table considers a different restriction of the model checking problem: the first column denotes if the labeling considered is (limit)

Robust	ϵ known	**MDP**	**Markov chain**
No	-	Undecidable	Open
Limit	No	Open	PSPACE
Yes	No	Open	PSPACE
Yes	Yes	EXPTIME	PSPACE

Table 9.1: Model checking complexity for distribution based semantics

	MDP	**Markov chain**
Limit Robustness	Undecidable	coNP-complete
Robustness	Undecidable	$\mathsf{coNP}^{\mathsf{RP}}$

Table 9.2: Complexity of Checking Robustness

robust or not, and the second column denotes whether the degree of robustness is given as an input. The results for MDPs are derived from those for PFAs as we saw in Section 7.2. In contrast the lower complexity of PSPACE for Markov chains is what we finally proved in Theorem 9.1. We also note that there are several problems that remain open in this space. The model checking problem for Markov chains against binary labeling (not necessarily limit robust) is open. The model checking problem for MDPs when the binary label is given to be robust but the degree of robustness ϵ is not known is also an open problem. We showed in Chapter 8 that the problem of checking (limit) robustness is tractable for Markov chains and their complexities are mentioned in Table 9.2.

Chapter 10

Conclusion

We conclude this thesis by summarizing the work, highlighting the results we have established and mentioning the main open problems. The focus of this thesis is on the model checking of probabilistic systems. Model checking involves determining whether a mathematical model of system satisfies a certain specification. We are specifically interested in a model called Markov Decision Processes (MDPs) which evolve, probabilistically and non-deterministically, over discrete time steps on a given finite state space. We looked at two different ways of assigning semantics to the model checking problem, execution based and distribution based.

The execution based semantics is defined with respect to a scheduler that can resolve non-determinism in the MDP. A scheduler induces a probability distribution on the executions of a MDP. We are interested in finding out if the probability of the specified set of executions is greater than a certain given threshold. This is called the quantitative model checking problem, the special version of it when the threshold is zero is called qualitative model checking problem. The specifications we consider are formulae in Linear Temporal Logic (LTL) over boolean propositions that annotate the states of the MDP. In order to solve the model checking problem we follow what is called an automata-theoretic approach which has two steps. The first step involves translating the LTL into an automaton and the second step involves analyzing the product of the MDP and the automaton. Since the size of the automaton plays a direct role in the complexity of this approach it is essential to obtain an efficient translation. With this in mind researchers have spent large efforts in translating LTL to deterministic automata which can be used for quantitative model checking. But it turns out that for qualitative model checking it is enough to consider limit deterministic automata. In this thesis we provide the first translation of large fragments of LTL to exponential sized

limit deterministic automata that can be used to speed up the qualitative model checking algorithm for those fragments (Chapter 4). We presented a tool called `Büchifier` that builds automata for LTL formula using this translation (Chapter 6). We also considered the quantitative model checking problem using existing deterministic automata translations, and proposed a new algorithm for analyzing the product of the automata and the MDP. This new space efficient algorithm helped us in determining the exact complexity of the quantitative problem for various LTL fragments (Chapter 5). Apart from these we also considered the complexity of translating the safety fragment of LTL to probabilistic monitors that can be used for model checking Markov chains and runtime monitoring (Chapter 3). One of the main open questions that we are left with is regarding our translation from LTL to limit deterministic automata; we presented a translation that was exponential for a large fragment called $\mathrm{LTL}_{\mathrm{D}}$, but the question remains whether this can be extended to a richer fragment and thus improve the qualitative model checking result.

In the distribution based semantics we considered Markovian schedulers; these are schedulers which resolve non-determinism by following a fixed sequence of actions. Such schedulers induce a sequence of probability distributions over the states of the MDP. We looked at propositions defined over such distributions, whose truth was derived by comparing the probability in a set of states against a given constant. The specifications we considered are ω-regular languages defined over those propositions. The model checking problem then became one of determining whether there is a Markovian scheduler which induces a sequence of probability distributions whose trace belongs to the given language. The problem happens to be undecidable in general hence we consider a restriction of the problem; we consider propositions which are (limit) robust w.r.t the MDP, i.e, their truth does not change on slight perturbation of the constant involved in the inequality. For this restricted case it turns out the the model checking problem becomes decidable when the degree of robustness is known. But the problem remains open when the degree of robustness is not given as an input. The problem of determining whether a proposition is robust happens to be undecidable. We then reconsider the same problem for Markov chains (which are MDPs without non-determinism). Our result here is that determining robustness for Markov chains is decidable (Chapter 8) is decidable, and the model checking problem against ω-regular specifications over robust

proposition is in PSPACE (Chapter 9). The question of model checking a Markov chain without the robustness assumption is one that is not settled, unlike in the case of MDPs where it is known to be undecidable.

References

[1] Büchifier. publish.illinois.edu/buchifier/.

[2] Eric Allender, Peter Bürgisser, Johan Kjeldgaard-Pedersen, and Peter Bro Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 38(5):1987–2006, 2009.

[3] Rajeev Alur and Salvatore La Torre. Deterministic generators and games for LTL fragments. *ACM Trans. Comput. Logic*, 5(1):1–25, January 2004. ISSN 1529-3785.

[4] Rajeev Alur, Salvatore La Torre, and P Madhusudan. Playing games with boxes and diamonds. *Lecture notes in computer science*, pages 128–143, 2003.

[5] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[6] T. Babiak, F. Blahoudek, M. Kretínský, and J. Strejeck. Effective translation of LTL to deterministic Rabin automata: Beyond the (F,G)-fragment. In *ATVA*, pages 24–39, 2013.

[7] Christel Baier and Marcus Größer. Recognizing omega-regular languages with probabilistic automata. In *LICS*, pages 137–146, 2005.

[8] Christel Baier, Stefan Kiefer, Joachim Klein, Sascha Klüppelholz, David Müller, and James Worrell. Markov chains and unambiguous büchi automata. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*, pages 23–42, 2016. doi: 10.1007/978-3-319-41528-4_2.

[9] A. Bertoni. The solution of problems relative to probabilistic automata in the frame of formal language theory. In *Proceedings of the 4th GI Jahrestagung*, volume 26 of *Lecture Notes in Computer Science*, pages 107–112, 1974.

[10] J. Richard Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960. ISSN 1521-3870. doi: 10.1002/malq.19600060105. URL http://dx.doi.org/10.1002/malq.19600060105.

[11] R. Chadha, A. P Sistla, and M. Viswanathan. Probabilistic automata with isolated cut-points. In *MFCS*, volume 8087 of *Lecture Notes in Computer Science*, pages 254–265. Springer, 2013.

[12] Rohit Chadha, A. Prasad Sistla, and Mahesh Viswanathan. On the expressiveness and complexity of randomization in finite state monitors. *J. ACM*, 56(5):26:1–26:44, August 2009. ISSN 0004-5411.

[13] Rohit Chadha, Vijay Anand Korthikanti, Mahesh Viswanathan, Gul Agha, and Young-Min Kwon. Model checking mdps with a unique compact invariant set of distributions. In *QEST*, pages 121–130, 2011.

[14] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263. ISSN 0164-0925. doi: 10.1145/5397.5399.

[15] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, UK, 1982. Springer-Verlag. ISBN 3-540-11212-X. URL http://dl.acm.org/citation.cfm?id=648063.747438.

[16] Edmund M. Clarke and E. Allen Emerson. *Design and synthesis of synchronization skeletons using branching time temporal logic*, pages 52–71. Springer Berlin Heidelberg, Berlin, Heidelberg, 1982. ISBN 978-3-540-39047-3. doi: 10.1007/BFb0025774. URL http://dx.doi.org/10.1007/BFb0025774.

[17] Anne Condon and Richard J. Lipton. On the complexity of space bounded interactive proofs (extended abstract). In *30th Annual Symposium on Foundations of Computer Science*, pages 462–467, 1989.

[18] Costas Courcoubetis and Mihalis Yannakakis. *Markov decision processes and regular events*, pages 336–349. Springer Berlin Heidelberg, Berlin, Heidelberg, 1990. ISBN 978-3-540-47159-2. doi: 10.1007/BFb0032043. URL http://dx.doi.org/10.1007/BFb0032043.

[19] Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995.

[20] E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "Not never" revisited: On branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, January 1986. ISSN 0004-5411. doi: 10.1145/4904.4999. URL http://doi.acm.org/10.1145/4904.4999.

[21] E. Allen Emerson and Chin-Laung Lei. Modalities for model checking: Branching time logic strikes back. *Sci. Comput. Program.*, 8(3):275–306, June 1987. ISSN 0167-6423. doi: 10.1016/0167-6423(87)90036-0. URL http://dx.doi.org/10.1016/0167-6423(87)90036-0.

[22] Javier Esparza and Jan Kretínský. From LTL to deterministic automata: A Safraless compositional approach. In *CAV*, pages 192–208, 2014.

[23] F.R. Gantmacher. *Applications Of The Theory Of Matrices*. Dover Books on Mathematics Series. Dover, 2005. ISBN 9780486445540.

[24] Marc Geilen. On the construction of monitors for temporal logic properties. *Electr. Notes Theor. Comput. Sci.*, 55(2):181–199, 2001. doi: 10.1016/S1571-0661(04)00252-X. URL http://dx.doi.org/10.1016/S1571-0661(04)00252-X.

[25] H. Gimbert and Y. Oualhadj. Probabilistic automata on finite words: Decidable and undecidable problems. In *Proceedings of the International Colloquium on Automata, Languages and Programming*, pages 527–538, 2010.

[26] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata Logics, and Infinite Games: A Guide to Current Research*. Springer-Verlag New York, Inc., New York, NY, USA, 2002. ISBN 3-540-00388-6.

[27] Gregor Gramlich. Probabilistic and nondeterministic unary automata. In *28th Internation Symposium on Mathematical Foundations of Computer Science*, pages 460–469, 2003.

[28] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994. ISSN 1433-299X. doi: 10.1007/BF01211866. URL http://dx.doi.org/10.1007/BF01211866.

[29] D. Kini and M. Viswanathan. Limit deterministic and probabilistic automata for LTL\GU. In *Proceedings of TACAS*, pages 628–642, 2015.

[30] Dileep Kini and Mahesh Viswanathan. Probabilistic Büchi automata for LTL\GU. Technical Report http://hdl.handle.net/2142/72686, University of Illinois at Urbana-Champaign, 2015.

[31] J. Klein and C. Baier. Experiments with deterministic ω-automata for formulas of linear temporal logic. *Theoretical Computer Science*, 363(2):182–195, 2006.

[32] Zuzana Komárková and Jan Kretínský. Rabinizer 3: Safraless translation of LTL to small deterministic automata. In *ATVA*, pages 235–241, 2014.

[33] Vijay Anand Korthikanti, Mahesh Viswanathan, Gul Agha, and YoungMin Kwon. Reasoning about mdps as transformers of probability distributions. In *QEST*, pages 199–208, 2010.

[34] I. Kremer, N. Nisan, and D. Ron. On randomized one-round communication complexity. *Symposium on Theory of Computing*, June 1995. doi: 10.1007/s000370050018.

[35] J. Kretínský and J. Esparza. Deterministic automata for the (F,G)-fragment of LTL. In *CAV*, pages 7–22, 2012.

[36] Orna Kupferman and Adin Rosenberg. The blow-up in translating LTL to deterministic automata. In *Proceedings of the 6th International Conference on Model Checking and Artificial Intelligence*, MoChArt'10, pages 85–94, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 978-3-642-20673-3.

[37] Orna Kupferman and MY Vardi. Model checking of safety properties. *CAV*, pages 1–27, 1999.

[38] E. Kushilevtiz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1996.

[39] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, pages 585–591, 2011.

[40] YoungMin Kwon and Gul Agha. Linear inequality ltl (iltl): A model checker for discrete time markov chains. In Jim Davies, Wolfram Schulte, and Michael Barnett, editors, *ICFEM*, pages 194–208, 2004.

[41] Z. Manna and A. Pnueli. *Temporal verification of reactive and concurrent systems: Specification*. Springer-Verlag, 1992.

[42] A. Morgenstern and K. Schneider. From LTL to symbolically represented deterministic automata. In *VMCAI*, pages 279–293, 2008.

[43] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995. ISBN 0-521-47465-5, 9780521474658.

[44] David E. Muller. Infinite sequences and finite machines. In *Proceedings of the 1963 Proceedings of the Fourth Annual Symposium on Switching Circuit Theory and Logical Design*, SWCT '63, pages 3–16, Washington, DC, USA, 1963. IEEE Computer Society. doi: 10.1109/SWCT.1963.8. URL http://dx.doi.org/10.1109/SWCT.1963.8.

[45] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. In *Verification, Model Checking, and Abstract Interpretation, 7th International Conference, VMCAI 2006*, pages 364–380, 2006. doi: 10.1007/11609773_24.

[46] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society. doi: 10.1109/SFCS.1977.32. URL http://dx.doi.org/10.1109/SFCS.1977.32.

[47] J. P. Queille and J. Sifakis. *Specification and verification of concurrent systems in CESAR*, pages 337–351. Springer Berlin Heidelberg, Berlin, Heidelberg, 1982. ISBN 978-3-540-39184-5. doi: 10.1007/3-540-11494-7_22.

[48] M.O. Rabin. Probabilitic automata. *Information and Control*, 6(3):230–245, 1963.

[49] Arnold Schönhage. On the power of random access machines. In *ICALP*, pages 520–529, 1979.

[50] E. Senata. *Non-negative Matrices and Markov Chains*. George Allen & Unwin Ltd, 1973.

[51] Leon Shargel and Andrew BC. Applied biopharmaceutics and pharmacokinetics. *Journal of Pharmaceutical Sciences*, 1985.

[52] Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínskỳ. Limit-deterministic Büchi automata for linear temporal logic. CAV, 2016.

[53] Abraham Silvers, Robert S Swenson, John W Farquhar, and Gerald M Reaven. Derivation of a three compartment model describing disappearance of plasma insulin-131i in man. *Journal of Clinical Investigation*, 48(8):1461, 1969.

[54] A. Prasad Sistla. Safety, liveness and fairness in temporal logic. In *Formal Aspect of Computing*, pages 495–511, 1999.

[55] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *Proc. of the 5th Ann. ACM Symposium on Theory of Computing,*, pages 1–9, 1973.

[56] Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16 (2):264–280, 1971.

[57] M. Vardi, P. Wolper, and A. P. Sistla. Reasoning about infinite computation paths. In *FOCS*, 1983.

[58] Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, SFCS '85, pages 327–338, Washington, DC, USA, 1985. IEEE Computer Society. ISBN 0-8186-0844-4. doi: 10.1109/SFCS.1985.12. URL http://dx.doi.org/10.1109/SFCS.1985.12.

[59] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency. LNCS, vol. 1043*, pages 238–266. Springer-Verlag, 1996.

[60] Klaus W. Wagner. Some observations on the connection between counting and recursion. *Theoretical Computer Science*, 47(3):131–147, 1986.

[61] A.C. Yao. Some complexity questions related to distributed computing. In *Proceedings of the ACM Symposium on Theory of Computation*, pages 209–213, 1979.

Appendix A

Details of LDBA Construction

A.1 Proof of Proposition 4.1

It is easy to see using propositional calculus that at least one of the three formula should hold, i.e $\neg\mathbf{F}\varphi \vee (\neg\mathbf{G}\varphi \wedge \mathbf{F}\varphi) \vee \mathbf{G}\varphi \equiv \mathbf{tt}$. Conjunction of every pair of them is false. This can be proved using propositional calculus for $\neg\mathbf{F}\varphi \wedge (\neg\mathbf{G}\varphi \wedge \mathbf{F}\varphi)$, and for $(\neg\mathbf{G}\varphi \wedge \mathbf{F}\varphi) \wedge \mathbf{G}\varphi$. It is easy see that $\neg\mathbf{F}\varphi \wedge \mathbf{G}\varphi$ is false since $\neg\mathbf{F}\varphi$ says φ can never hold, while $\mathbf{G}\varphi$ asserts that φ is always true, which is a contradiction. If φ is of the form $\mathbf{F}\psi$ then $w \vDash \varphi$ iff $w \nvDash \neg\mathbf{F}\varphi$.

A.2 Derivative

In Section 4.2 we introduced the concept of derivative by providing a declarative definition. Here we will elaborate on the syntactic definition of the concept.

We describe some terminology related to normal form representation for temporal formulae. A *term* t is a conjunction of formulae $\varphi_0, .., \varphi_k$ denoted as a set $t = \{\varphi_0, .., \varphi_k\}$. A term over φ is a term in which all formulae are subformulae of φ or their dependents (formulae $\mathbf{F}\psi$ for internal $\phi\,\mathcal{U}\,\psi$ and and $\mathbf{X}(\phi\,\mathcal{U}\,\psi)$ for all $\phi\,\mathcal{U}\,\psi$). The set of all terms over φ is denoted by $\mathcal{T}(\varphi)$. A *form* is a disjunction of a finitely many terms $t_1, .., t_n$ represented as $\langle t_1, .., t_n\rangle$. A form is said to be over φ if every term in it is over φ. Set of all such forms is denoted by $\mathcal{F}(\varphi)$. A single term t can also be interpreted as the form $\langle t\rangle$ depending on the context it is used. We use $\langle\psi\rangle$ to denote the form with a single term $\{\psi\}$. False is denoted by the empty form $\langle\rangle$ and true is represented as $\langle\emptyset\rangle$. If there are two terms t_i, t_j in a form such that $t_i \subseteq t_j$ then we can drop t_j because t_j implies t_i. For a set of terms T, let $\min(T)$

be the form consisting of the minimal (according to the subset relation) terms in T. Let $\nu_1 \cup \nu_2$ be the set of terms contained in either ν_1 or ν_2. The join of two forms ν_1, ν_2 denoted by $\nu_1 \sqcup \nu_2$ is the form $\min(\nu_1 \cup \nu_2)$. The meet of two forms ν_1, ν_2 denoted by $\nu_1 \sqcap \nu_2$ is the form $\min(\{t_1 \cup t_2 \mid t_1 \in \nu_1, t_2 \in \nu_2\})$.

We say a term t is *ex-free* if $t \cap \Lambda_\varphi$ is empty, and form f is ex-free if each term in it is ex-free. Next, we introduce the concept of *consistency*.

Definition A.1 (Consistency). *A term e is said to be locally consistent if:*

- $\phi \vee \psi \in e$ *then* $\phi \in e$ *or* $\psi \in e$.
- $\phi \wedge \psi \in e$ *then* $\phi \in e$ *and* $\psi \in e$
- $(\phi \mathcal{U} \psi \in e$ *and* $\phi \mathcal{U} \psi \notin \Lambda_\varphi)$ *then* $\mathbf{F}\psi \in e$
- $\phi \mathcal{U} \psi \in e$ *then either* $(\phi \in e$ *and* $\mathbf{X}(\phi \mathcal{U} \psi) \in e)$ *or* $\psi \in e$
- $\mathbf{ff} \notin e$

A term e is said to be consistent with input symbol $\sigma \in 2^P$ if:

- *if* $p \in e$ *then* $p \in \sigma$
- *if* $\neg p \in e$ *then* $p \notin \sigma$

A term $e \in \mathcal{T}(\varphi)$ is said to be consistent with an FG-prediction $\pi \in \Pi(\varphi)$ if:

- $\mathbf{F}\psi \in e$ *then* $\mathbf{F}\psi \notin \alpha(\pi)$
- $\mathbf{G}\psi \in e$ *then* $\mathbf{G}\psi \in \alpha(\pi)$

A term $e \in \mathcal{T}(\varphi)$ is said to be consistent with ex-choice $\lambda \subseteq \Lambda_\varphi$ if:

- $\forall\, \phi \mathcal{U} \psi \in e \cap \Lambda_\varphi$*:* $\psi \in e$ *iff* $\phi \mathcal{U} \psi \in \lambda$
- $\forall\, \phi \vee \psi \in e \cap \Lambda_\varphi$*:* $\phi \in e$ *iff* $\phi \vee \psi \in \lambda$

The notion of local consistency is an extension of the concept of "local informativeness" introduced in [24]. A term is locally consistent if every compound formula of the form $(\wedge / \vee / \mathcal{U})$ present in the term is appropriately supported by the presence of its immediate subformulae/dependents. A term that is locally consistent gives a *proof* for the satisfaction

of each compound formula present in it. The proof is local in the sense that it gives a way to satisfy the current and not any future obligations that need to be met. Consider $\phi \mathcal{U} \psi$, for which to be satisfied one needs:

- ψ to hold at some point (which is expressed by the presence of dependent $\mathbf{F}\psi$), and
- either ψ holds now (presence of ψ) or ϕ holds now and $\phi \mathcal{U} \psi$ holds at the next step (presence of ϕ, $\mathbf{X}(\phi \mathcal{U} \psi)$)

For $\wedge/\vee$ we need both/one of the arguments present for the term to be locally consistent. Note that local consistency does not handle literals because the current input σ is supposed to tell us their truth; and those requirements are encoded in σ consistency constraints. Similarly π tells us the truth of $\mathbf{F}$,$\mathbf{G}$-formulae which are encoded in π consistency constraints. An ex-choice $\lambda \in \Lambda_\varphi$ also dictates additional constraints. The ex-choice λ dictates how each external $\mathcal{U}/\vee$ subformula is satisfied if it needs to be. If $\phi \mathcal{U} \psi \in \lambda$ then it must be immediately satisfied by the presence of ψ. A $\phi \vee \psi \in \lambda$ must be satisfied by the presence of ϕ. λ provides us with a resolution of choices created by external $\mathcal{U}/\vee$.

For notational simplicity, we are going to combine the three forms of constraints (input symbol, FG-prediction, ex-choice) into an extended symbol:

Definition A.2 (Extended symbol). *An extended symbol for an LTL formula φ over propositions P is a triple $(\sigma, \pi, \lambda) \in 2^P \times \Pi(\varphi) \times \Lambda_\varphi$. We will use $\mathcal{E}_\varphi$ denote the space $2^P \times \Pi(\varphi) \times \Lambda_\varphi$. We say a term t is consistent with $\varepsilon \in \mathcal{E}_\varphi$ if t is consistent with each component of ε.*

(We will also sometimes refer to the pair (σ, π) as an extended symbol, this is useful when the ex-choice becomes irrelevant)

For a sequence of symbols ρ (finite or infinite) over 2^P, an *extension* is an equally long sequence $w = \{(\rho_i, \pi_i, \lambda_i)\}$ over $\mathcal{E}_\varphi$.

Next, we define the *expansion* of a term w.r.t an extended input. The expansion is a form consisting of terms that describe different ways to satisfy the given term.

Definition A.3 (Expansion). *For $t \in \mathcal{T}(\varphi)$ and $\varepsilon \in \mathcal{E}_\varphi$, the expansion $\mathcal{X}(t, \varepsilon)$ is the form $\min(T)$ where T is the set of all terms e such that $t \subseteq e$, e is locally consistent, and consistent with ε. Given form ν we define the expansion $\mathcal{X}(\nu, \varepsilon)$ as $\bigsqcup_{t \in \nu} \mathcal{X}(t, \varepsilon)$*

The successor of a term t is the term consisting of all the temporal obligations that are pending in t:

Definition A.4 (Successor). *The successor of a term t denoted by $\mathcal{S}(t)$ is defined as the term $\{\psi \mid \mathbf{X}(\psi) \in t\}$. For a form f, $\mathcal{S}(f)$ is defined as the form* $\min(\{\mathcal{S}(t) \mid t \in f\})$.

Next, we define the *derivative*. Here we directly define what corresponds to the weakest derivative as in Definition 4.6 and simply refer to it as the derivative. Given a form f, a finite sequence of input symbols ρ and an extension w of ρ: the derivate denoted by $\nabla(f, w)$ corresponds to the obligation such that if any infinite continuation ρ' satisfies it then it guarantess f to be true at $\rho\rho'$ given that w is the prefix of a *sound* extension for $\rho\rho'$. Informally, an extension is sound if the guesses made by the FG-predictions and ex-choices along the word are correct. The derivative can be seen as a generalization and a declarative version of the various unfolding operations [35, 22].

Definition A.5 (Derivative). *For a form $f \in \mathcal{F}$, and extended symbol $\varepsilon \in \mathcal{E}_\varphi$ define the derivate $\nabla(f, \varepsilon)$, as the form $\mathcal{S}(\mathcal{X}(f, \varepsilon))$. We extend the definition to finite words over $\mathcal{E}_\varphi$ as: $\nabla(f, \epsilon) = f$ and $\nabla(f, \varepsilon w) = \nabla(\nabla(f, \varepsilon), w)$. For a given extension w we shall use $\nabla_i^j(f)$ as a shorthand for $\nabla(f, w[i, j])$*

Observe that the derivative of a form only consits of $\mathcal{U}$ subformulae and arguments of $\mathbf{X}$ subformulae of the given form due to the application of $\mathcal{S}$.

Now we are ready to describe the automata construction using this definition. A single state in our construction is a 4-tuple (μ, ν, π, m) where μ and ν are forms over φ, π is a FG-prediction and m is a counter. Note that forms are just a different way of representing formulae and will be more convenient when proving correctness and efficiency of the construction. The only operations a form needs to emulate are the $\wedge/\vee$ which are done by $\sqcap/\sqcup$. The construction is exactly as in Definition 4.7 with forms replacing formulae and the analogous boolean operations for forms.

Definition A.6 (Construction). *Given a formula $\varphi \in$ LTL over propositions P, let $\mathcal{D}(\varphi)$ be the NBA (Q, δ, I, F) over the alphabet 2^P defined as follows:*

- *Q is the set $\mathcal{F}(\varphi) \times \mathcal{F}(\varphi) \times \Pi(\varphi) \times [n]$ where $n = |\mathbb{F}(\mathcal{C}(\varphi))| + 1$*

- δ *is the set of all transitions of the form* $(\mu, \nu, \pi, m) \xrightarrow{\sigma} (\mu', \nu', \pi', m')$ *such that*

  (a) $\alpha(\pi) \subseteq \alpha(\pi')$ *and* $\gamma(\pi) = \gamma(\pi')$

  (b) $\mu' = \nabla(\mu \sqcap \langle t \rangle, \varepsilon)$ *for some* $\lambda \subseteq \Lambda_\varphi$

  where $t = \{\psi \mid \mathbf{F}\psi \in \beta(\pi) \cap \alpha(\pi') \text{ or } \mathbf{G}\psi \in \alpha(\pi)\}$ *and* $\varepsilon = (\sigma, \pi, \lambda)$

  (c) $m' = \begin{cases} (m+1) \pmod{|\mathbb{F}(\gamma)| + 1} & \nu = \langle \emptyset \rangle \\ m & \text{otherwise} \end{cases}$

  (d) $\nu' = \begin{cases} \langle \psi_{m'} \rangle & \nu = \langle \emptyset \rangle \\ \nabla(\nu, (\sigma, \pi)) \sqcup \langle \psi_m \rangle & \text{otherwise} \end{cases}$

  where $\{\mathbf{F}\psi_0, \ldots, \mathbf{F}\psi_k\}$ *is an enumeration of* $\mathbb{F}(\gamma)$, $\psi_0 = \mathbf{tt}$

- I *is all states of the form* $(\langle \varphi \rangle, \langle \emptyset \rangle, \pi, 0)$

- F *is all states of the form* $(\mu, \langle \emptyset \rangle, \pi, 0)$ *where* $\beta(\pi) = \emptyset$, $\mu \neq \langle\rangle$, μ *is ex-free*

A.3 Proof of Correctness

In order to prove correctness (Theorem 4.2) we define an annotation for an accepting run. The annotation is a sequence of pairs (u, v) where u and v represent expansions of terms in μ and ν respectively that are true:

Definition A.7 (Annotation). *Given an accepting run of automaton* $\mathcal{D}(\varphi)$ *over word* ρ

$$(\mu_0, \nu_0, \pi_0, m_0) \xrightarrow{\rho_0} \cdots (\mu_i, \nu_i, \pi_i, m_i) \xrightarrow{\rho_i} \cdots \qquad (\star)$$

an annotation is a (finite/infinite) sequence of pairs $(u_i, v_i) \in \mathcal{T} \times (\mathcal{T} \cup \{\mathsf{null}\})$ *where:*

$$\forall i \geq 0: \quad \lambda_i \subseteq \Lambda_\varphi \text{ such that } \mu_{i+1} = \nabla(\mu_i \sqcap \Psi_i, (\rho_i, \pi_i, \lambda_i)) \tag{A.1}$$

$$\forall i \geq 0: \quad u_i \in \mathcal{X}\big(t \sqcap \Psi_i,\ \rho_i, \pi_i, \lambda_i)\big) \text{ where } t = (\mathcal{S}(u_{i-1}) \ \texttt{if}\ \ i > 0 \ \texttt{else}\{\varphi\})$$

$$\text{where} \quad \Psi_i = \{\psi \mid \mathbf{F}\psi \in \beta(\pi_i) \cap \alpha(\pi_{i+1}) \text{ or } \mathbf{G}\psi \in \alpha(\pi_i)\} \tag{A.2}$$

$$\forall i \geq 0: \quad \emptyset \in \nu_i \iff v_i = \emptyset \tag{A.3}$$

$$\forall i > 0: \quad v_{i-1} = \emptyset/\mathsf{null} \implies v_i = \mathsf{null} \text{ or } v_i \in \mathcal{X}\big(\langle\psi_{m_i}\rangle, (\rho_i, \pi_i)\big) \tag{A.4}$$

$$\forall i > 0: \quad v_{i-1} \neq \emptyset/\mathsf{null} \implies v_i \in \mathcal{X}(\mathcal{S}(v_{i-1}), (\rho_i, \pi_i)) \tag{A.5}$$

Lemma A.1. *For a given accepting run of* $\mathcal{D}(\varphi)$ $(\star)$

$$\forall n \geq 0;\ \forall s \in \mu_n;\ \forall x \in \mathcal{X}(s \sqcap \Psi_n, (\rho_n, \pi_n, \lambda_n));\ \forall y \in \mathcal{X}(\nu_n, (\rho_n, \pi_n))$$

there exists an annotation of length $n{+}1$ *ending in* (x, y)

Proof. First note that the requirements for sequence of u_i and those for v_i in an annotation are independent. We show the existence of a sequence $\{u_i\}_{0 \leq i \leq n}$ ending in x and sequence $\{v_i\}_{0 \leq i \leq n}$ ending in y each satisfying their respective conditions. Combining the sequences will then gives us the required annotation. First note that λ_i in condition A.1 is given by the transtions taken in the accepting run. We perform indunction on n to construct the sequences.

Base Case: when $n = 0$ we have $\mu_n = \langle\varphi\rangle$ (initial state condition), so s can only be $\{\varphi\}$ and hence $x \in \mathcal{X}(\{\varphi\} \sqcap \Psi_0, (\rho_0, \pi_0, \lambda_n))$ which satisfies condition A.2. For $n = 0$ we have $\nu_n = \langle\emptyset\rangle$, so $y = \emptyset$ which would satisfy condition A.3. Conditions A.4 and A.5 do not apply for the base case (see quantification on the conditions).

Inductive Case: We assume the statement is true for n upto i and prove it for $n = i + 1$.

Extending u: Let t be a term in μ_i such that the quantified s is contained in $\nabla(t \sqcap \Psi_i, (\rho_i, \pi_i, \lambda_i)) = \mathcal{S}(\mathcal{X}(t \sqcap \Psi_i, (\rho_i, \pi_i, \lambda_i)))$. Such a term t should exist by the fact that $s \in \mu_{i+1}$ and how μ_{i+1} is derived from μ_i in Rule **(b)**. Now let x' be the term in $\mathcal{X}(t \sqcap \Psi_i, (\rho_i, \pi_i, \lambda_i))$ for which $\mathcal{S}(x') = s$. By the inductive hypothesis we have $\{u_j\}_{0 \leq j \leq i}$ that ends in x' which

satisfies the condition A.2. We append x to this sequence to obtain the required sequence of length $i+1$

Extending v: Consider the case when $\nu_i = \langle\emptyset\rangle$, here we know $\nu_{i+1} = \langle\psi_{m_{i+1}}\rangle$ implying $y \in \mathcal{X}(\langle\psi_{m_{i+1}}\rangle, (\rho_{i+1}, \pi_{i+1}))$. Condition A.3 is satisfied for $v_{n+1} = y$ because y is non-empty, as $\psi_{n_{i+1}}$ is contained in y. For $\nu_i = \langle\emptyset\rangle$ we need to check condition A.4, which is satisfied because $y \in \mathcal{X}(\langle\psi_{m_{i+1}}\rangle, (\rho_{i+1}, \pi_{i+1}))$. Next, consider the case when $\nu_i \neq \langle\emptyset\rangle$, here we know $\nu_{i+1} = \nabla(\nu_i, (\rho_i, \pi_i)) \sqcup \langle\psi_{m_i}\rangle$. Now suppose r be the term in ν_{i+1} for which $y \in \mathcal{X}(r, (\rho_{i+1}, \pi_{i+1}))$. If $r \in \nabla(\nu_i, (\rho_i, \pi_i))(= \mathcal{S}(\mathcal{X}(\nu_i, (\rho_i, \pi_i))))$ let $y' \in \mathcal{X}(\nu_i, (\rho_i, \pi_i))$ for which $r = \mathcal{S}(y')$. By the induction hypothesis we have $\{v_j\}_{0\leq j\leq i}$ that ends in y'. We append y to this sequence to obtain the required sequence of length $i+1$. If $r = \{\psi_{m_i}\}$, consider the last index k where $n_k \neq n_i$. (If such a k does not exist, then all m_i have to be zero in which case we have $v_i = \emptyset$ for all i). Here ν_k has to be $\langle\emptyset\rangle$ owing to the fact that counter changes only when ν becomes $\langle\emptyset\rangle$. By induction hypothesis there exists a sequence $\{v_j\}_{0\leq j\leq k}$ ending in $\emptyset$. Appending $\mathsf{null}^{(k-i-1)}y$ to this gives us the required sequence. □

Corollary A.1. *Every accepting run of $\mathcal{D}(\varphi)$ has an infinite annotation*

Proof. Note that in an accepting run $\mu_i \neq \emptyset$ for any i, which means there will always exist $s \in \mu_i$ such that $\mathcal{X}(s \cup \Psi_i, (\rho_i, \pi_i, \lambda_i)) \neq \emptyset$ (otherwise $\mu_{i+1} = \emptyset$). In an accepting run we also know that $\nu_i = \langle\emptyset\rangle$ for infinitely many i. Hence using Lemma (A.1) we have finite sized annotations of arbitrarily large lengths. Note that every prefix of an annotation is also a valid annotation. We can arrange all these annotations and their prefixes in an infinite rooted tree where the root is the empty annotation, and there is an edge between two annotations if one of them is a prefix of the other obtained by removing the last element. Every node in this tree has finite degree because the space of each element in this sequence is finite $(\mathcal{T} \times (\mathcal{T} \cup \{\mathsf{null}\}))$. Finally we use König's lemma to obtain an infinite path in this tree which gives us the required infinite annotation. □

Let $Set : \mathcal{T} \cup \{\mathsf{null}\} \to \mathcal{T}$ be the function which maps every $\mathcal{T}$ to itself and maps null to the empty term $\emptyset$.

Lemma A.2. *For an accepting run $(\star)$ with an infinite annotation $\{(u_i, v_i)\}_{i\geq 0}$, it is the case that $\forall i \geq 0 \;\; \forall\, \theta \in (u_i \cup Set(v_i)) : \rho(i) \vDash \theta$*

Proof. We perform induction on the size of θ. We shall use the fact that $S_i = u_i \cup Set(v_i)$ is a term that is locally consistent and consistent with π_i and w_i.

For the base case when θ is a literal $p/\neg p$ it has to be the case that $p/\neg p$ is respectively true for $\rho(i)$ because it is contained in a term that is consistent with w_i.

If θ is of the form $\phi \vee/\wedge \psi$ then using local consistency we get that either (or both) of ϕ or ψ is present in S_i, and hence true at $\rho(i)$ due to the inductive hypothesis which gives us the truth of θ at $\rho(i)$.

If θ is $\phi \mathcal{U} \psi$: if $\theta \notin \Lambda_\varphi$ then local consistency tells us that $\mathbf{F}\psi \in S_i$ and using the induction hypothesis we get that $\mathbf{F}\psi$ is true at $\rho(i)$. Let $j \geq i$ the smallest index such that $\rho(j) \vDash \psi$. $\forall k : i \leq k < j$ we can inductively prove (this is a separate induction on k) that $\phi, \mathbf{X}(\phi \mathcal{U} \psi) \in S_k$ using the facts: that ψ is absent from all such S_k (induction hypothesis), local consistency of S_k and that $\mathbf{X}(\phi \mathcal{U} \psi)$ transfers $\phi \mathcal{U} \psi$ to the next step (definition of annotation and $\mathcal{S}$). If $\theta \in \Lambda_\varphi$ then there is a $j \geq i$ such that θ is absent in μ_j (because once a final state is reached μ becomes ex-free), picking the smallest such j one obtains the point at which ψ is true which can be proved by induction on j like we did in previous case.

If θ is $\mathbf{F}\psi$ then using the fact that S_i is π_i consistent we can infer that $\mathbf{F}\psi \in \beta(\pi_i) \cup \gamma(\pi_i)$. If $\psi \in \beta(\pi_i)$ then $\exists j \geq i$ such that $\mathbf{F}\psi \in \beta(\pi_j) \cap \alpha(\pi_{j+1})$ (due to **(a)**). Now we know $\psi \in \Psi_j$ and hence $\psi \in S_j$ (from definition of annotation). Using the induction hypothesis we get $\rho(j) \vDash \psi$ and hence $\rho(i) \vDash \mathbf{F}\psi$. If $\psi \in \gamma(\pi_i)$ then we know that the counter will eventually (say at $j \geq i$) become the index m_j corresponding to ψ in γ (γ doesn't change along a run, see **(a)**). Let j be the smallest such index. Consider the smallest $k \geq j$ for which $v_k \neq \mathsf{null}$. m_k has to be equal to m_j because the counter cannot change while v_i is empty. From property (A.4) of annotations we get that $v_k \in \mathcal{X}(\{\psi\}, \pi_k, w_k)$. This tells us that $\psi \in S_k$ and by the induction hypothesis we have $\rho(k) \vDash \psi$ thus proving $\rho(i) \vDash \mathbf{F}\psi$.

If θ is $\mathbf{G}\psi$ then using the fact that S_i is consistent with π_i we infer that $\mathbf{G}\psi \in \alpha(\pi_i)$. Now using **(a)** we infer that $\mathbf{G}\psi \in \alpha(\pi_j)$ for all $j \geq i$. Using property (A.2) of annotations we get that $\psi \in u_j (\subseteq S_j)$. Applying the induction hypothesis we get that $\rho(j) \vDash \psi$ for all $j \geq i$ and hence $\rho(i) \vDash \mathbf{G}\psi$.

□

Corollary A.2. *If w has an accepting run in $\mathcal{D}(\varphi)$ then $w \vDash \varphi$.*

Proposition A.1. *If $w \vDash \varphi$ then w has an accepting run in $\mathcal{D}(\varphi)$*

Proof. Define the run $(\mu_i, \nu_i, \pi_i, m_i)$ as follows. Let π_i be such that

$$\alpha(\pi_i) = \{\mathbf{G}\psi \in \mathcal{C}(\varphi) \mid \rho(i) \vDash \mathbf{G}\psi\} \cup \{\mathbf{F}\psi \in \mathcal{C}(\varphi) \mid \rho(i) \nvDash \mathbf{F}\psi\}$$
$$\gamma(\pi_i) = \{\mathbf{G}\psi \in \mathcal{C}(\varphi) \mid \rho(i) \nvDash \mathbf{FG}\psi\} \cup \{\mathbf{F}\psi \in \mathcal{C}(\varphi) \mid \rho(i) \vDash \mathbf{GF}\psi\}$$

Define λ_i as $\{\, \psi \,\mathcal{U}\, \phi,\, \phi \vee \psi \in \Lambda_\varphi \mid \rho(i) \vDash \phi\}$. Fixing the sequences π_i, λ_i resolves all the non-determinism present along a run, we can then define μ_i, ν_i, m_i as described by the initial state and the transition relation. We will be then left to check that the run is indeed an accepting one.

We first show that $\forall i\, \mu_i \neq \langle\rangle$. We prove a stronger statement $\exists\, t \in \mu_n,\, \forall\, \psi \in t : \rho(n) \vDash t$ by induction on n. Base case is trivial as $\mu_0 = \langle\varphi\rangle$ for the initial state. Next consider the inductive case where $n = i + 1$. Let t_i be term in μ_i which is true according to the induction hypothesis. We can then recursively construct the set $e \supseteq (t_i \cup \Psi_i)$ such that all formulae we add to e are true by looking at the truth of the immediate subformulae of the formulae present in e. For example consider $\phi\,\mathcal{U}\,\psi \in e' = (e \cap \Lambda_\varphi)$, if $\phi\,\mathcal{U}\,\psi \in \lambda_i$ then we add ψ to e as we know that $\rho(i) \vDash \psi$ by definition of λ_i, otherwise we add $\phi, \mathbf{X}(\phi\,\mathcal{U}\,\psi)$ to e to ensure local consistency. Note that the formulae we add will be true at $\rho(i)$. Whenever there is a choice as to how we can make a certain formula true then we non-deterministically pick one. We do so for every formulae in e. The set e constructed will be locally consistent, consistent with π_i (by it's definition and the fact that all formulae in e are true at $\rho(i)$), consistent with w_i (by the fact all formulae in e hold at $\rho(i)$) and λ_i consistent (due to the way we construct e). Consider all minimal such e (among all those that can be constructed this way) and they would have to be present in $\mathcal{X}(t \cup \Psi_i, (w_i, \pi_i, \lambda_i))$. Then pick t_{i+1} as $\mathcal{S}(e)$ which is a minimal among all such e (to ensure $t_{i+1} \in \mu_{i+1}$), and observe that every formulae in $\mathcal{S}(e)$ is true at $\rho(i{+}1)$ by semantics of $\mathbf{X}$ and the fact that all formulae in e hold at $\rho(i)$.

Next we show that ν_i is $\langle\emptyset\rangle$ for infinitely many i. In order to do so we define the metric f which for a given word ρ and formula ψ gives us a number $f(\rho, \psi)$ which gives us an upper bound on the number of steps it takes ν to become $\langle\emptyset\rangle$ along ρ if it were to start at $\langle\psi\rangle$.

Definition A.8. *For a formula φ and a word ρ define $f(\rho, \varphi) \in \mathbb{N} \cup \{\infty\}$ such that: $f(\rho, \varphi) =$*

∞ *if* $\rho \not\models \varphi$ *otherwise we recursively define it as follows*

$$
\begin{aligned}
f(\rho, \varphi) &= 1 && \text{if } \varphi \text{ is } p/\neg p/\mathbf{F}\psi/\mathbf{G}\psi \\
f(\rho, \mathbf{X}\phi) &= 1 + f(\rho(1), \phi) \\
f(\rho, \phi \wedge \psi) &= \max(f(\rho, \phi), f(\rho, \psi)) \\
f(\rho, \phi \vee \psi) &= \min(f(\rho, \phi), f(\rho, \psi)) \\
f(\rho, \phi \,\mathcal{U}\, \psi) &= \max_{j<i}(j + f(\rho(j), \phi), i + f(\rho(i), \psi)) && i \text{ is min s.t } \rho(i) \models \psi
\end{aligned}
$$

We also extend this definition for a term t*: define* $f(\rho, t)$ *as* $\max_{\varphi \in t} f(\rho, \varphi)$

Next, using the above metric we show the following statement which claims that if a term t is true at $\rho(j)$ then taking the derivative of t for sufficiently many times will yield the $\emptyset$ term.

$$
\forall j \, \forall t \, \forall k \; (\forall \psi \in t : \rho(j) \models \psi) \wedge (k = j + f(\rho(j), t)) \Rightarrow \emptyset \in \nabla(\{t\}, \pi(j,k), w(j,k)) \tag{A.6}
$$

We prove this using induction on k. Consider a term t that holds for $\rho(i)$. One can show that $\exists t' \in \nabla(t, (\rho_i, \pi_i))$ such that t' is true for $\rho(i+1)$ and $f(\rho(i+1), t') < f(\rho(i), t)$. Using this along with the induction hypothesis will prove the inductive case. For the base case when $f(\rho(i), t)$ is 1 the formulae in t can only be boolean combinations of $p/\neg p/\mathbf{F}\psi/\mathbf{G}\psi$ which do not produce any obligations ($\mathbf{X}$ formulae) which implies that $\emptyset \in \nabla(t, (\rho_i, \pi_i))$.

Now, for contradiction assume ν is $\langle\emptyset\rangle$ finitely many times, and let i be the index succeeding the last point where $\nu = \langle\emptyset\rangle$. Let ψ denote the formula ψ_{m_i}. Since $\mathbf{F}\psi \in \gamma(\pi_i)$ we know that ψ holds infinitely often by definition of $\gamma(\pi_i)$, so consider a point $j \geq i$ such that $\rho(j) \models \psi$. Using (A.6) we get that that within $f(\rho(j), \psi)$ steps ν would have to be $\langle\emptyset\rangle$ which is a contradiction.

Next, we note that $\beta(\pi_i)$ is empty for sufficiently large i. In order to prove μ eventually becomes ex-free we can once again use a metric similar to f to argue that within finite steps every external subformula disappears from μ. □

Proposition A.2. $\mathcal{D}(\varphi)$ *is limit deterministic*

Proof. First note that ν' and n are deterministically updated, they only depend on ν, π which is a part of the state and σ which is the current input symbol. The only non-determinism in

the automaton comes from the evolution of μ and π. In a final state, the π cannot change any further due to monotonicity and $\beta(\pi)$ being empty. In a final state μ becomes ex-free and remains ex-free from then on because the formula introduced in μ come from Ψ all of which are internal. This implies that the ex-choice λ does not play a role in determining μ (as it is ex-free) and hence μ is also updated deterministically from then on. □

A.4 Efficiency

In order to prove Theorem 4.3, we first observe identities about the derivative that we shall use in proofs appearing later in this Section.

Lemma A.3. *For forms A and B and extended symbol ε:* $\nabla(A \sqcup / \sqcap B, \varepsilon) = \nabla(A, \varepsilon) \sqcup / \sqcap \nabla(B, \varepsilon)$

Lemma A.4. *Given $\varphi \in \mathrm{LTL}\backslash GU$, a subformula ψ, and an extended input symbol $\varepsilon \in \mathcal{E}_\varphi$, the derivative of ψ, $\nabla(\psi)$ (short for $\nabla(\psi, \varepsilon)$) satisfies the following identities depending on the structure of ψ:*

$$\nabla(\psi_1 \,\mathcal{U}\, \psi_2) = \begin{cases} \nabla(\psi_2) \sqcup (\nabla(\psi_1) \sqcap \langle \psi_1 \,\mathcal{U}\, \psi_2 \rangle) & \text{if } \psi_1 \,\mathcal{U}\, \psi_2 \notin \Lambda_\varphi \\ \nabla(\psi_2) & \text{if } \psi_1 \,\mathcal{U}\, \psi_2 \in \lambda \\ \nabla(\psi_1) \sqcap \langle \psi_1 \,\mathcal{U}\, \psi_2 \rangle & \text{otherwise} \end{cases}$$

$$\nabla(\psi_1 \vee \psi_2) = \begin{cases} \nabla(\psi_1) \sqcup \nabla(\psi_2) & \text{if } \psi_1 \vee \psi_2 \notin \Lambda_\varphi \\ \nabla(\psi_1) & \text{if } \psi_1 \vee \psi_2 \in \lambda \\ \nabla(\psi_2) & \text{otherwise} \end{cases}$$

$$\nabla(\psi_1 \wedge \psi_2) = \nabla(\psi_1) \sqcap \nabla(\psi_2)$$

$$\nabla(\mathbf{F}\psi) = \langle\rangle \quad \text{if } \mathbf{F}\psi \in \alpha(\pi) \text{ else } \langle\emptyset\rangle \qquad \nabla(p) = \langle\emptyset\rangle \text{ if } p \in \sigma \text{ else } \langle\rangle$$

$$\nabla(\mathbf{G}\psi) = \langle\emptyset\rangle \text{ if } \mathbf{G}\psi \in \alpha(\pi) \text{ else } \langle\rangle \qquad \nabla(\neg p) = \langle\rangle \text{ if } p \in \sigma \text{ else } \langle\emptyset\rangle$$

Next, we observe that the terms in the derivative of an $\mathrm{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$ formula φ w.r.t a word of length k consist only of subformulae at depth k in φ, and hence derivatives of φ

of order greater than h are either true or false where h is the height of φ. The lemma can be proved by induction on k.

Lemma A.5. *For $\varphi \in \mathrm{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$, $w \in \mathcal{E}^k_\varphi$, $t \in \nabla(\langle\varphi\rangle, w)$: every $\phi \in t$ is such that ϕ is a formula at depth k within φ.*
Corollary: If $\varphi \in \mathrm{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$ is of height $h \geq 0$, then $\nabla(\langle\varphi\rangle, w) \in \{\langle\rangle, \langle\emptyset\rangle\}$ for all $|w| > h$.

Now we see how to represent the space of reachable ν and μ. Let us fix a formula φ and an infinite sequence of extended symbols w. Note that the ν component of a run cycles through $\mathbb{F}(\gamma)$. When ν becomes $\langle\emptyset\rangle$ it moves to the next $\mathbf{F}\psi$ in $\mathbb{F}(\gamma)$. $\mathbb{F}(\gamma)$ is at most as large as the given formula, hence it suffices to show a bound on reachable ν for a single $\mathbf{F}\psi$. With this in mind we fix ψ and define ν_i inductively as follows: $\nu_0 = \langle\psi\rangle$ and $\nu_{i+1} = \nabla(\nu_i, w_i) \sqcup \langle\psi\rangle$. For μ define $\mu_0 = \langle\varphi\rangle$ and $\mu_{i+1} = \nabla(\mu_i \sqcap \Psi_i, w_i)$ where $\Psi_i = \{\psi \mid \mathbf{F}\psi \in \beta(\pi_i) \cap \alpha(\pi_{i+1}) \text{ or } \mathbf{G}\psi \in \alpha(\pi_i)\}$. The sequence μ_i describes the μ component of a run. Our aim is to find a representation for ν_i, μ_i and show that the number of different possible representations is exponential. The following Proposition proved using Lemma A.5 gives us a representation.

Proposition A.3. *If $\psi \in \mathrm{LTL}(\mathbf{F}, \mathbf{G}, \wedge, \vee)$ and $\varphi \in \mathrm{LTL}_\mathrm{D}$ are of height k, and $l = \max(i-k, 0)$*

$$\nu_i = \bigsqcup_{j=l}^{i} \nabla^i_j(\langle\psi\rangle) \text{ or } \langle\emptyset\rangle \qquad\qquad \mu_i = \bigsqcap_{j=l}^{i-1} \nabla^i_j(\Psi_j) \sqcap \nabla^i_0(\langle\varphi\rangle) \text{ or } \langle\rangle$$

Proof. Consider ν_i, we can prove that it is $\sqcup^i_{j=0} \nabla^i_j(\psi)$ by inducting on i and using Lemma A.3. Then we observe that the first $l-1$ elements are either true ($\langle\emptyset\rangle$) or false ($\langle\rangle$) due to Lemma A.5. The representation above follows immediately. For μ_i the structure can be derived in a similar fashion the only difference is the extra term $\nabla^i_0(\varphi)$ that arises due to the initial condition. □ □

Note if ν_i is not $\langle\emptyset\rangle$ then it is completely determined by the substring of extended symbols $w[l, i]$. There are at most $(2^{|P|}.3^{|\varphi|}.2^{|\Lambda_\varphi|})^k$ such substrings for each length and there are k different lengths. Hence we get that ν_i can take on at most exponentially different values.

Observe that in μ_i the part $\prod_{j=l}^{i-1} \nabla_j^i(\Psi_j)$ can take exponentially many different values once again due to the fact that it only depends on the $w[l,i]$. What remains to be seen that $\nabla_0^i(\langle\varphi\rangle)$ takes on at most exponentially different values. The next Lemma states that every derivative of φ over $w[0,i]$ can be expressed as the derivative of a single term t over $w[l,i]$.