Theory of
Hybrid Systems
Informatik 2

Master of Science Thesis

Minimal Critical Subsystems for PCTL

Amith Belur Nagabushana

Supervisor:
Dipl. Inform. Nils Jansen

Advisors:
Prof. Dr. Erika Ábrahám
Prof. Dr. Ir. Joost-Pieter Katoen, PDEng

May 2013

Abstract

Probabilistic counterexamples form a crucial part of model checking probabilistic systems represented as Markov chains. They provide invaluable debugging information. Minimal critical subsystems are known to be succinct representations of these counterexamples. The SAT-modulo theories formulation and the mixed integer linear programming formulation compute optimal counterexamples in form of minimal critical subsystems. We already have formulations for reachability properties and ω-regular properties. We extend these formulations to handle properties specified in probabilistic computation tree logic for discrete-time Markov chains. We evaluate these formulations and the optimizations applied with a number of experiments.

Erklärung

Hiermit versichere ich, dass ich die vorgelegte Arbeit selbstständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe. Alle benutzten Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate wurden als solche gekennzeichnet.

Amith Belur Nagabushana
Aachen, den 21. Mai 2012

Acknowledgements

This thesis has been a tremendous learning experience for me. I would not have been able to complete it without the support of many. I want to whole-heartedly thank Nils Jansen for his invaluable advice and his endless patience through this process. I would like to show my gratitude to Prof. Dr. Erika Ábrahám for providing me with an excellent opportunity to do interesting research. I want to thank Prof. Dr. Ir. Joost-Pieter Katoen for not only agreeing to be the co-advisor but also providing invaluable information through his book on model checking. I want to thank my parents, my sister and my friends for their continued support. Finally, I thank Arundhati Roy for providing inspiration.

Contents

Chapter 1

Introduction

There has been an increasing adoption of software in several industries such as the automotive industry, the aerospace industry, healthcare, the chemical plants and the nuclear plants to enumerate a few. Due to the direct interaction with humans and the environment, errors in software can interfere with the desirable behaviour having catastrophic effects. For instance, a software flaw in a radiation therapy machine claimed the lives of six cancer patients [BK08, p.2]. The safety and reliability issues caused by software errors necessitate robust verification techniques. The importance of formal methods in computer science is likened to the importance of applied mathematics in other fields of engineering. It has been an effective method to reduce large amounts of time invested in verification of complex systems and enabling early detection of defects [BK08]. Model-based verification is one such verification technique based on models describing the system in a mathematically precise manner. *Model checking* is a model-based verification technique that evolved in the context of concurrent program verification [Cla08]. Concurrency errors are notorious for their uncertainty. It is hard to catch the errors by program testing and difficult to reproduce them. These errors were detected using hand constructed proofs coupled with temporal logic until 1981 when Edmund Clarke and Allen Emerson made a case for viewing this as a state-exploration problem. They argued that the finite concurrent systems can be viewed as a *Kripke structure* and can be checked if they satisfy a desirable property specified in propositional temporal logic.

The model checking problem can be succinctly described in the following way.

> Let M be a Kripke structure (i.e., state-transition graph). Let f be a formula specified in temporal logic (i.e., the specification). Find all states s of M such that $M,s \models f$ [Cla08].

This translates to checking if the structure M is a *model* of the formula f. Model checking can be viewed as a three-phase process as described in [BK08]. The first phase is called the *modelling phase* that involves modelling the system under consideration and specifying the property to be checked. The second phase is the *running phase* that involves running the model checker. The third phase is the *analysis phase* that involves checking if the property in consideration is satisfied. If the property is violated, a violating behaviour or a *counterexample* is generated.

Generation of counterexamples is considered a crucial aspect of model checking. It not only serves as essential information for debugging complex systems, but also assists in generating models. Model checking can be seen as exploring all possible states of a given system. This leads to state explosion for problems of industrial complexity necessitating *abstraction-based methods.* Abstraction-based methods involve removing details of the system that are irrelevant to the property under consideration. One such abstraction is *over-approximation* that involves enriching a model by removing constraints. It allows for false counter-examples due to an incorrect approximation. Every instance of false counterexample leads to further refining of the model. This method of achieving a correct model through false counterexamples is called *counterexample-guided abstraction technique* [CGJ+03]. There has been lot of research on counterexample generation. Here are a few algorithms, [HWZ08, CV10, GMZ04, CGMZ95, CJLV02, BP12, SB05]. Let us consider an example to illustrate a counterexample for a property.

Example 1.0.1. *A Kripke structure $\mathcal{K}$ is a tuple $\mathcal{K} = (S, I, R, AP, L)$ where $S = \{s_1,s_2,s_3,s_4\}$ is the set of finite states, $s_1 \subseteq I$ is the set of initial states, $R \subseteq S \times S$ is a transition relation, $AP = \{a,b\}$ is a set of atomic propositions and $L : S \to 2^{AP}$ is a labelling function. Let us consider Kripke structure in Figure 1.1. s_1 is the initial state and following are the labels, $L(s_1) = a, L(s_2) = a, L(s_3) = b, L(s_4) = a$. We need to check if the property $\forall(a\mathcal{U}b)$ is violated. This property is specified in* computation tree logic *explained later (2.3.1). The property states that every path from a state s needs to reach a state where $\{b\}$ holds traversing through the states where $\{a\}$ holds. The only state where $\{b\}$ holds is s_3. We can see that s_4 has no path leading to s_3. A path leading to s_4 is a counterexample. Since there is a loop with s_1 and s_2 in the path leading to s_4, counterexamples can be of form $s_1s_2s_4$, $s_1s_2s_1s_2s_4$ and so forth. A single path in sufficient to show that $\mathcal{K} \not\models \forall(a\mathcal{U}b)$.*

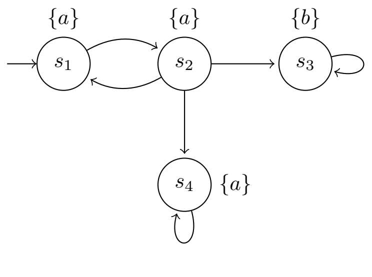


Figure 1.1: Kripke structure $\mathcal{K}$ checked for the property $\forall(a\mathcal{U}b)$

Probabilistic model checking is a variant of model checking that is used to verify probabilistic systems for constraints that are not hard or rigid. For instance, a non-rigid constraint is a constraint that need not always hold. It can hold utmost 99% of the time. These properties are required in systems that exhibit phenomena of stochastic nature. This includes randomized algorithms such as distributed algorithms that have randomness in their logic to break symmetry. For instance, if we have n indistinguishable processors in a ring that are tasked with electing a leader with a symmetric deterministic algorithm, it is shown that the election of a leader would be possible only with an infinite computation or an erroneous result. For termination with a correct result, we require a probabilistic algorithm. The leader election algorithms require each identical processor to have an independent random number generator to break the symmetry of every identical processor choosing the same number [IR90]. Systems which exhibit unreliable and unpredictable behaviour such as message delivery or processor failure also require non-determinism to be modelled. *Markov chains* are used to model these systems. The transitions in such a system are based on probability distributions over the states. Markov chains can be bifurcated into *Markov decision processes (MDPs)* and its deterministic simplification *discrete-time Markov chains (DTMCs)*. In an MDP, each state has multiple probability distributions of transitioning to its successors and a probability distribution is chosen non-deterministically. While a DTMC has only a single probability distribution at every state. We only deal with DTMCs in this work.

Probabilistic model checking deals with two kinds of properties, namely, *qualitative properties* and *quantitative properties*. Qualitative properties are the kinds of properties where a desirable event is expected to happen with certainty, i.e., with a probability 1 or an undesirable event is expected to never occur, i.e., occur with a probability 0. Quantitative properties, on the other hand, are expected to happen with an upper bound or lower bound on the probability. For a leader election algorithm, an example of a quantitative property would be, given 4 nodes, the probability that a leader would be elected within 50 steps is at least 0.5. The stress in this work is on quantitative properties. However, devising methods to model check and generate counterexamples for quantitative properties can be easily extended to factor in qualitative properties by setting appropriate probability bounds to be satisfied. Calculating probabilities of reaching certain states in DTMCs is reduced to solving linear equation systems [BK08]. A set of cases studies in PRISM demonstrate the applications of DTMC model checking [KNP$^+$12].

The challenge with probabilistic model checking is providing counterexamples or *diagnostic feedback* of the violation of the property in the case it is refuted. Unlike traditional model checking, a single path need not be sufficient as a counterexample. In case of probabilistic model checking, the properties in consideration have a probability bound that needs to be violated. Hence, it can be a set of paths with a total probability mass that violates a bound. [HKB09] provides an approach for calculating path-based counterexamples. It calculates the smallest counterexample, i.e., a counterexample that deviates the most from the required probability bound given that it has the smallest number of paths. Given an *until formula*, the problem of finding the smallest counterexample can be seen as a *k shortest paths* problem where the k shortest or most probable paths form the smallest counterexample. If the probability bound is a lower bound, negation of the property is considered converting it to an upper bound.

The same method is applied exploiting the duality of upper and lower probability bounds.

Example 1.0.2. *A Markov chain $\mathcal{M}$ is a tuple $\mathcal{M} = (S, I, P, AP, L)$ where $S = \{s_1,s_2,s_3,s_4\}$ is the set of finite states, $s_1 \subseteq I$ is the set of initial states, $P : S \times S \to [0,1] \subseteq \mathbb{R}$ is the transition probability function, $AP = \{a,b\}$ is the set of atomic propositions and $L : S \to 2^{AP}$ is the labelling function. Let us consider the Markov chain in Figure 1.2. The property in consideration is $P_{<0.5}[a\mathcal{U}b]$. This means we need to find out if the probability at the initial state s_1 of reaching a state where $\{b\}$ holds, which is s_3, through a set of states where $\{a\}$ holds, the remaining states, is less than 0.5. $\pi_1 = s_1s_2s_3$ is one such path. Probability (Pr) of reaching s_3 in π_1 can be calculated through the transition probabilities, i.e., $Pr(\pi_1) = 0.5 \times 0.25 = 0.125$. It is not enough to violate the property. Since we have a loop, we can consider $\pi_2 = s_1s_2s_1s_2s_3$. If we go on adding the probabilities, we get*

$$\sum_{i=1}^{\infty} Pr(\pi_i) = 0.25 \times 0.5 + 0.25 \times 0.5^2 \ldots = 0.5$$

Hence, a counterexample for the property $P_{<0.5}[a\mathcal{U}b]$ is a countably infinite set of paths.

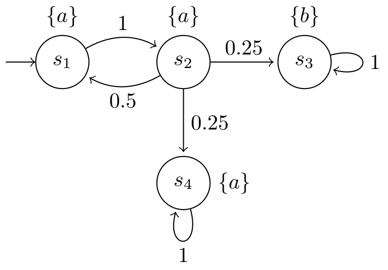


Figure 1.2: Markov chain $\mathcal{M}$ checked for the property $P_{<0.5}[a\mathcal{U}b]$

It has been observed that path-based examples can be doubly exponential in the problem size making it unusable for debugging. Hence, [HKB09] suggests succinct counterexamples in form of regular expressions. In the example 1.0.2, the counterexample can be represented as $(s_1s_2)^*s_3$.

Critical subsystems serve as an alternative to path-based counterexamples. A critical subsystem is a subset of states of a DTMC along with the corresponding transitions between these states allowing for sub-stochastic probability distributions. It is a compact representation of the path-based counterexamples easing the process of debugging. While the size of path-based counterexamples in the worst case can be doubly exponential, critical subsystems are bounded by the number of states in a DTMC. There have been several approaches for computing critical subsystems. [AL10] proposes a best first search while [JÁK$^+$11] proposes a technique based on a hierarchical SCC-based abstraction. However, experimental results show that the critical subsystems generated from those proposed methods are significantly larger than the minimum. [WJÁ$^+$12a, WJÁ$^+$12b] have addressed this issue proposing methods to generate minimal critical subsystems (MCS) where the minimality is either based on states or transitions. Let us illustrate an MCS with an example.

Example 1.0.3. *Consider the Example 1.0.2. While the path-based counterexample here is an infinite set of paths, an MCS as represented in Figure 1.3 is the minimum possible sub-DTMC that contributes to violating the probability threshold. This is because probability of s_1 and s_2 reaching s_3 is* 0.5*. Removing any more states would make s_3 unreachable from the initial state s_1.*

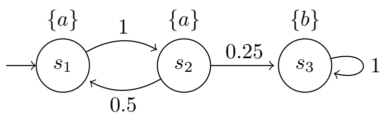


Figure 1.3: MCS as a counterexample for the property $P_{<0.5}[a\mathcal{U}b]$

The computation of minimal critical subsystems is seen as a minimization problem in quantifier-free linear arithmetic [WJÁ+12b]. There are two formulations to the problem. The first formulation is SAT-modulo theories (SMT) formulation that allows disjunctive constraints within the linear program. The second formulation is a mixed integer linear program (MILP) that uses only conjunctions. The MILP formulation is known to outperform SMT. All the work on MCS until now has been focused on reachability properties and a general class of ω-regular properties. This is the first work that extends this formulation to compute MCS for properties specified in the complete *probabilistic computation tree logic (PCTL)* (see 2.3.2). This includes constrained reachability, bounded constrained reachability, reachability to a nested PCTL formula constrained by another nested PCTL formula. E.g., $P_{<0.5}[P_{>=0.5}[a\mathcal{U}b]\mathcal{U}b]$. Such a nested formula would require that we construct a parse tree of sub-formulae and do a bottom up traversal calculating the satisfiability sets for each sub-formula. After building satisfiability sets, based on whether the outer most formula is satisfied by the initial states of the DTMC, we can check if the property is refuted by the DTMC. In case of a refutation, we can come up with the minimum critical subsystem that refutes the probability bound. Let us consider an example.

Example 1.0.4. *Let us consider the DTMC in Figure 1.2 and check if it refutes $P_{<0.5}[P_{>=0.5}[a\mathcal{U}b]\mathcal{U}b]$. Based on the computation of MCS from Figure 1.3 in Example 1.0.3, we can assume that the probabilities of states s_1,s_2,s_3 of reaching s_3 are at least* 0.5*. Hence, all three states satisfy the inner formula $\Phi' \equiv P_{>=0.5}[a\mathcal{U}b]$. The property in consideration is $P_{<0.5}[\Phi'\mathcal{U}b]$. This means we need to check the probability of reaching s_3 from the states satisfying Φ', i.e., s_1,s_2,s_3 to s_3. This would be the same as calculating the probabilities within the sub-DTMC shown in Figure 1.3.*

The work computes an MCS by recursive computation of satisfiability sets of all the sub-formulae of the PCTL property in consideration. This is done by building a linear constraint system. If the property is violated, the linear program builds an objective function with the goal of building a state-minimal subsystem violating the probability threshold subject to the constraints. Besides the SMT and MILP formulation for full PCTL, we provide optimizations to speed up this process of finding the solution. We explain how the constraints are built and explain efficacy of the optimizations by applying them to randomized protocols as a part of the experimental evaluation.

Chapter 2

Foundations

In this chapter, we introduce concepts of probability theory and probabilistic systems. We discuss Markov chains which are used to model the probabilistic systems. We define the branching tree logic with which we specify the properties of interest for probabilistic systems. We discuss model checking and methods of generating counterexamples. We later delve into minimal critical subsystems and the various methods of computing them.

2.1 Probability Theory

Probability theory provides a mathematical model to describe experiments with random outcomes. To ensure that the permissible collection of subsets of these random outcomes are closed under basic operations, we use a *σ-algebra* to describe the set of events.

Definition 2.1.1 (σ-Algebra of Sets). *σ-algebra is a pair $(E,\mathfrak{E})$ where E is a non-empty set and $\mathfrak{E}$ is a subset of the power set of E that contains the empty set and is closed under the complementation and countable unions, i.e.,*

- $\emptyset \in \mathfrak{E}$
- *if* $e \in \mathfrak{E}$, *then* $\overline{e} = E \setminus e \in \mathfrak{E}$
- *if* $e_1, e_2, \ldots \in \mathfrak{E}$, *then* $\displaystyle\bigcup_{n \geq 1} e_i \in \mathfrak{E}$

If all subsets $e_i \subseteq \mathfrak{E}$ are countable, then the union $\bigcup_{i \geq 0} e_i$ is also countable. The biggest σ-algebra over E is if $\mathfrak{E} = 2^E$. The smallest σ-algebra over E is if $\mathfrak{E} = \{\emptyset, E\}$ [BK08, 754].

Definition 2.1.2 (Probability Measure)**.** *A probability measure is a function $Pr : \mathfrak{E} \to [0,1] \subseteq \mathbb{R}$. It is a measure on $\mathfrak{E}$ normalized to 1 such that if $(e_n)_{n\geq 1}$ is a family of pairwise disjoint events $e_n \in \mathfrak{E}$, then:*

$$Pr(\bigcup_{n\geq 1} e_n) = \sum_{n\geq 1} Pr(e_n).$$

Let $e \in \mathfrak{E}$ be an event. A probability measure of e is given by $Pr(e) = \frac{|e|}{|E|}$. Here $|e|$ and $|E|$ are the cardinalities of the sets. This implies $Pr(\emptyset) = 0$ and $P(E) = 1$.

Example 2.1.1. *Let E be the numbers on a dice, i.e., $E = \{1,2,3,4,5,6\}$. Let e_1 be the event of rolling a prime number, i.e., $e_1 = \{2,3,5\}$. $Pr(e_1) = \frac{|e_1|}{|E|} = \frac{1}{2}$.*

Definition 2.1.3 (Probability Space)**.** *A triple $(E,\mathfrak{E},Pr)$ where E is a non-empty set, $\mathfrak{E}$ is a σ-algebra on E and Pr is a measure on $\mathfrak{E}$ with $Pr(E) = 1$ forms a probability space.*

Since a probability space contains a probability measure and a σ-algebra, it is non-negative, the empty set is contained in it and it exhibits countable additivity.

2.2 Markov Chains

Markov Chains are transition systems with probabilistic choices. *Discrete-Time Markov Chains* have only probabilistic choices. However, certain behaviour such as interleaving of concurrent processes requires non-determinism. *Markov Decision Processes* have a combination of non-determinism and probabilistic behaviour where a certain probability distribution at every state is chosen non-deterministically. We only deal into Discrete-Time Markov Chains here.

Definition 2.2.1 (Discrete-time Markov Chains)**.** *A discrete-time Markov chain (DTMC) is a tuple $\mathcal{M} = (S,P,\iota_{init},AP,L)$ where*

- *S is a countable or finite non-empty set of states*
- *$P : S \times S \to [0,1] \subseteq \mathbb{R}$ is a transition probability function such that $\forall s \in S, \sum_{s'\in S} P(s,s') \leq 1$*
- *$\iota_{init} : S \to [0,1]$ is the initial distribution, such that $\sum_{s\in S} \iota_{init}(s) = 1$*
- *AP is the set of atomic propositions*
- *$L : S \to 2^{AP}$ is a labelling function.*

$P(s,s')$ gives the probability of a transition from s to s' in a single step. The condition on the transition probability allows for sub-stochastic distributions. This is required for defining critical subsystems (see 2.5.1). Usually, the sum of probabilities is required to be 1. A DTMC $\mathcal{M}$ with sub-stochastic distributions P can be transformed into a DTMC $\mathcal{M}'$ with stochastic distributions P' by introducing a sink state denoted by $s_\perp$ where $P'(s,s') = P(s,s')$ for all $s,s' \in S$, $P'(s_\perp,s_\perp) = 1$ and $P'(s,s_\perp) = 1 - P(s,S)$ and $P'(s_\perp,s) = 0$ for all $s \in S$.[WJÁ+12a]. For algorithmic purposes, we assume probability values are rational [BK08].

Remark. *A thing to note is a Markov chain loses its memory of its starting state after a sufficiently large number of transitions. This means the probability of being at a given step n with a large n is not dependent on its initial state [Nel95, p.334]. Markov chain is said to be* memoryless *and at any given state, the set of states preceding it do not matter. This is called the* Markov property.

Definition 2.2.2 (Initial States). *Let $\mathcal{M} = (S,P,\iota_{init},AP,L)$ be a DTMC. A state $s \in S$ is an initial state if $\iota_{init}(s) > 0$ denoted by set $I = \{s \in S | \iota_{init}(s) > 0\}$.*

Definition 2.2.3 (Absorbing State). *A state $s \in S$ is an absorbing state if $P(s,s) = 1$ and $P(s,t) = 0$ for any other state $t \in S \setminus \{s\}$*

Example 2.2.1. *Figure 2.1 illustrates a DTMC $\mathcal{M}_1$ with $\iota_{init}(s_1) = 1$ and $AP = \{a\}$. s_4 is the only state with a label, $L(s_4) = a$. States s_4 and s_6 are the absorbing states.*

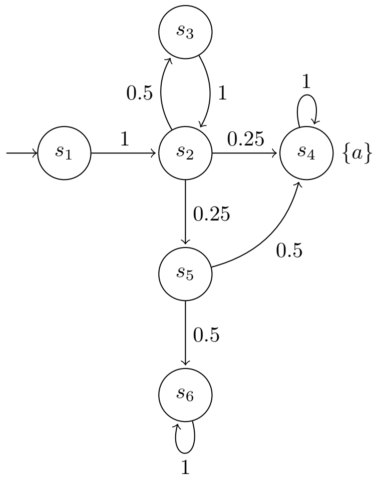


Figure 2.1: DTMC $\mathcal{M}_1$

Definition 2.2.4 (Path Fragment). *A finite path fragment $\hat{\pi}$ of a DTMC $\mathcal{M} = (S,P,\iota_{init},AP,L)$ is a finite sequence of states $s_0 s_1 \ldots s_n$ such that $s_i \in succ(s_{i-1})$ where $succ(s_{i-1}) = \{s_i \in S \mid P(s_{i-1},s_i) > 0\}$ and $0 \leq i \leq n$, $n \geq 0$. An infinite path fragment π is an infinite state sequence $s_0 s_1 \ldots$ such that $s_i \in succ(s_{i-1})$ for all $i \geq 0$ [BK08].*

Definition 2.2.5 (Path)**.** *A path π of DTMC $\mathcal{M} = (S,P,\iota_{init},AP,L)$ is an infinite path fragment and $inf(\pi)$ denotes the set of states that are visited infinitely often.* $\inf(\pi)$ *is non-empty for a finite DTMC [BK08].*

Remark. *Strictly speaking a path needs to be infinite. However, we are only interested in finite path fragments which is referred to as paths for simplicity. We will refer to finite path fragments as paths from here. If $\pi = s_0 s_1 \ldots$, then $\pi[i]$ refers to the $i + 1st$ state.*

Definition 2.2.6 (Trace)**.** *A trace of a finite fragment $\hat{\pi} = s_0 s_1 \ldots s_n$ is defined as $trace(\hat{\pi}) = L(s_0)L(s_1)\ldots L(s_n)$. A trace of an infinite path fragment $\pi = s_0 s_1 \ldots$ is defined as $trace(\pi) = L(s_0)L(s_1)\ldots$ [BK08].*

Definition 2.2.7. *$Paths(\mathcal{M})$ of a DTMC $\mathcal{M} = (S,P,\iota_{init},AP,L)$ is a set of infinite sequences $s_0 s_1 s_2 \ldots \in S^\omega$ with $P(s_i,s_{i+1}) > 0$ for $i \geq 0$. $Paths_{fin}(\mathcal{M})$ is a set of finite path fragments $\hat{\pi} = s_0 s_1 s_2 \cdots s_n \in Paths_{fin}(\mathcal{M})$ with $P(s_i,s_{i+1}) > 0$ for $0 \leq i \leq n$ [BK08].*

Probability Measure of a Markov Chain In order to define probabilities for events within a a DTMC $\mathcal{M} = (S,P,\iota_{init},AP,L)$, we associate a probability space $(E,\mathfrak{C},Pr)$ with $\mathcal{M}$ where $E = Paths(\mathcal{M})$. The σ-algebra associated with the probability space comprising the pair $(\mathfrak{C}, Pr)$ can be generated using *cylinder sets.*

Definition 2.2.8 (Cylinder Sets)**.** *A cylinder set $Cyl(\hat{\pi})$ where $\hat{\pi} = s_0 s_1 s_2 \cdots s_n \in Paths_{fin}(\mathcal{M})$ is a set of infinite paths π which have $\hat{\pi}$ as a finite prefix.*

Definition 2.2.9 (σ-algebra for a Markov chain)**.** *The σ-algebra of a DTMC $\mathcal{M}$ is the smallest σ-algebra that contains all cylinder sets $Cyl(\hat{\pi})$ where $\hat{\pi}$ ranges over all finite path fragments in $\mathcal{M}$.*

Remark. *Probability measure $Pr^{\mathcal{M}}$ on σ-algebra $\mathfrak{C}^{\mathcal{M}}$ is given by*

$$Pr^{\mathcal{M}}(Cyl(s_0 \cdots s_n)) = \iota_{init}(s_0) \cdot P(s_0 \cdots s_n)$$

where

$$P(s_0 \cdots s_n) = \prod_{0 \leq i \leq n} P(s_i,s_{i+1}).$$

If the length of the path fragment is zero, then $P(s_0) = 1$. For paths starting in a non-initial state s, we apply the same formula on $\mathcal{M}_s$. Here, $\mathcal{M}_s$ is a DTMC obtained from modifying $\mathcal{M}$ by making s the unique initial state. The probability measure on the modified DTMC is denoted by $Pr^{\mathcal{M}_s}$ or $Pr^{\mathcal{M}}_s$ [BK08].

2.3 Probabilistic Computation Tree Logic

The probabilistic computation tree logic (PCTL) is a branching tree logic used to specify properties for probabilistic systems. In order to explain PCTL, we need an understanding of *computation tree logic* (CTL) since PCTL is an extension to CTL. CTL is based on the notion of a branching temporal logic yielding an infinite tree of states as opposed to a linear notion of time. CTL has the flexibility to specify if all branches originating from a state exhibit a desired behaviour or if it is only a subset of branches that satisfy this desired behaviour.

Computation Tree Logic Computation tree logic or CTL has a two stage syntax with the formulae bifurcated into state and path formulae. State formulae are assertions of atomic propositions in the states and their branching structure. While path formulae are assertions of temporal behaviour of paths. Temporal operators such as the $\bigcirc$ operator, called the next-step operator, and the $\mathcal{U}$ operator, called the until operator, cannot be combined with boolean connectives. A temporal operator needs to be preceded by the $\forall$ operator or the $\exists$ operator.

Definition 2.3.1 (Syntax of CTL)**.** *Grammar for PCTL state formula*

$$\Phi := \; tt \mid a \mid \neg\Phi \mid \Phi_a \wedge \Phi_b \mid \exists[\varphi] \mid \forall[\varphi]$$

where tt is the true literal, $a \in AP$. $J \in [0{,}1] \subseteq \mathbb{R}$. J is a permissible interval for the probability measure at a state. φ is a path formula with the grammar

$$\varphi := \; \bigcirc\Phi \mid \Phi_a \, \mathcal{U} \, \Phi_b$$

where Φ,Φ_a,Φ_b are state formulae and $n \in \mathbb{N}$.

Let us discuss the path formulae. The next path formula denoted by $\bigcirc\Phi$ is a formula that requires the next state, after a transition of one step from the current state, to satisfy the state formula Φ. The until path formula denoted by $\Phi_a \cup \Phi_b$ specifies constrained reachability. It requires a path to traverse through Φ_a-states until the target state Φ_b is reached. The $\forall$ operator or the $\exists$ operator require to be followed by a path formula. For a state s to satisfy the $\forall$ operator, it requires that all paths from the state s satisfy the path formula φ. For a state s to satisfy the $\exists$ operator, it requires that at least one path from the state s satisfies the path formula φ. Let us now compare the syntax of CTL with the syntax of PCTL.

Definition 2.3.2 (Syntax of PCTL)**.** *Grammar for PCTL state formula*

$$\Phi := \; tt \mid a \mid \neg\Phi \mid \Phi_a \wedge \Phi_b \mid P_J[\varphi]$$

where tt is the true literal, $a \in AP$. $J \in [0{,}1] \subseteq \mathbb{R}$. J is a permissible interval for the probability measure at a state. φ is a path formula with the grammar

$$\varphi := \; \bigcirc\Phi \mid \Phi_a \, \mathcal{U} \, \Phi_b \mid \Phi_a \, \mathcal{U}^{\leq n} \, \Phi_b$$

where Φ,Φ_a,Φ_b are state formulae and $n \in \mathbb{N}$

Comparing the syntax of CTL with PCTL, the $\forall$ operator and the $\exists$ operator are replaced by P_J. This provides PCTL with a flexibility to define *quantitative* along with *qualitative* properties. Qualitative properties are the kinds of properties where a desirable event is expected to happen with certainty, i.e., with a probability 1 or an undesirable event is expected to never occur, i.e., with a probability 0. Quantitative properties, on the other hand, are expected to happen with an upper bound or lower bound on the probability. A legal PCTL formula requires that a path formula φ is immediately preceded by the P operator. For convenience, an interval J can also be expressed as an upper or lower bound. E.g., $P_{[0,0.5]}[\varphi] = P_{\leq 0.5}[\varphi]$. The bound can also be a strict inequality. E.g., $P_{[0,0.5)}[\varphi] = P_{<0.5}[\varphi]$.

The additional path formula in PCTL is $\Phi_a \mathcal{U}^{\leq n} \Phi_b$, also called the bounded until formula, requires Φ_b to be satisfied within $n > 0$ steps traversing through Φ_a-states. This is explained in detail under the satisfaction relation for PCTL (see 2.3.4). Other operators can be derived from the operators specified in PCTL grammar. $\Phi_a \vee \Phi_b$ is equivalent to $\neg(\neg\Phi_a \wedge \neg\Phi_b)$. The *eventually* formula $\Diamond\Phi$ is equivalent to the until formula $tt\mathcal{U}\Phi$. The $\Box$ operator called the *always* operator can be derived from a combination of the duality of eventually operator and probability bounds.

$$P_{\leq\lambda}[\Box\Phi] \quad\equiv\quad P_{>\lambda}[\Diamond\neg\Phi]$$

The eventually formula $\Diamond\Phi$ is used to specify safety critical properties. Φ can be seen as an undesirable event. The safety critical property would specify that the probability of reaching such a Φ-state should be less than a certain probability bound. Φ-states are referred to as the *target states*. We need to calculate the *reachability probabilities* for such states to check if a specification is satisfied. In order to devise methods for calculating reachability probabilities, we need to introduce *relevant paths* and *relevant states.*

Definition 2.3.3. *Let $\mathcal{M} = (S,P,\iota_{init},AP,L)$ be a DTMC with* target states *$T \subseteq S$. A path $\pi = s_0 s_1 s_2 \ldots s_n$ is called a* relevant path *if $s_0 \in I$, $s_i \notin T$ for $0 \leq i < n$, $s_n \in T$. A set of states $S^{rel}_{\mathcal{M}}$ is called a* set of relevant states *if every state in the set is on the relevant path [WJÁ+12a].*

Reachability Probabilities In order to calculate the probability of reaching a certain state where $a \in AP$ holds, i.e., $\Diamond a$, we perform a reachability analysis. This helps us to compute the relevant states with respect to the set of target states $T = \{s_i \subseteq S \mid L(s_i) = a\}$. The probability of reaching an a-state can be calculated by the following linear equation system [BK08].

$$p_s = \begin{cases} 1 & \text{if } a \in L(s) \\ 0 & \text{if } s \notin S^{rel}_{\mathcal{M}} \\ \sum_{s' \in succ(s)} P(s,s') \cdot p_{s'} & \text{if } s \in S^{rel}_{\mathcal{M}} \end{cases} \tag{2.1}$$

Remark. *The states that are not relevant can be removed from the DTMC without altering reachability probabilities.*

Example 2.3.1. *Given the DTMC $\mathcal{M}_1 = (S,P,\iota_{init},AP,L)$ in Figure 2.1, let us calculate the probability measure $Pr^{s_i}_{\mathcal{M}}(\Diamond a)$ where $s_i \in S$. We have $a \in L(s_4)$.*

s_i	p_{s_i}
s_1	0.75
s_2	0.75
s_3	0.75
s_4	1
s_5	0.5
s_6	0

Table 2.1: Reachability probabilities for $\Diamond a$ in Figure 2.1

We need to check for the states that can be reached from the initial state s_1 and have a path to s_4. We can see that $S^{rel}_{\mathcal{M}} = \{s_1,s_2,s_3,s_4,s_5\}$ are the reachable states from both s_1 and s_4. Therefore, they are on the relevant path. Let p_s be a variable that holds the reachability probability where $s \in S$. We can assign $p_{s_6} = 0$ since $s_6 \notin S^{rel}_{\mathcal{M}}$. $p_{s_4} = 1$ since $a \in L(s_4)$. The remaining states are assigned probabilities based on the formula $p_s = \sum_{s' \in succ(s)} P(s,s') \cdot p_{s'}$. Table 2.1 has all the probabilities calculated.

Definition 2.3.4 (Satisfaction Relation for PCTL). *We look into the conditions necessary for various PCTL formulae to be satisfied for a state or a path within a DTMC $\mathcal{M} = (S,P,\iota_{init},AP,L)$. Let $s \in S$, $a \in AP$, Φ,Φ_a and Φ_b be PCTL formulae. The satisfaction relation is defined accordingly,*

$$\begin{aligned} s &\models a && \textit{iff} && a \in L(s) \\ s &\models \neg\Phi && \textit{iff} && s \not\models \Phi \\ s &\models \Phi_a \wedge \Phi_b && \textit{iff} && s \models \Phi_a \textit{ and } s \models \Phi_b \\ s &\models P_J[\varphi] && \textit{iff} && Pr(s \models \varphi) \in J \end{aligned}$$

Here, for a DTMC $\mathcal{M}$, $Pr^{\mathcal{M}}(s \models \varphi) = Pr^{\mathcal{M}}_s\{\pi \in Paths(s) \mid \pi \models \varphi\}$. Given a path π, where $\pi = s_1 s_2 s_3 \ldots$ and $\pi[i]$ is $i+1st$ state. The satisfaction relation for path formulae,

$$\begin{aligned} \pi &\models \bigcirc\Phi && \textit{iff} && \pi[1] \models \Phi \\ \pi &\models \Phi_a \mathcal{U} \Phi_b && \textit{iff} && \exists j \geq 0 : \pi[j] \models \Phi_b \wedge (\forall 0 \leq i < j : \pi[i] \models \Phi_a) \\ \pi &\models \Phi_a \mathcal{U}^{\leq n} \Phi_b && \textit{iff} && \exists 0 \leq j \leq n : \pi[j] \models \Phi_b \wedge (\forall 0 \leq i < j : \pi[i] \models \Phi_a) \end{aligned}$$

Definition 2.3.5 (PCTL Model Checking). *Given a PCTL property $P_{\leq\lambda}[\varphi]$, PCTL Model Checking checks if the probability mass of satisfying φ at all initial states of a DTMC $\mathcal{M}$ put together exceeds the bound λ.*

Remark. *All DTMCs we consider have a* unique initial state *s_I. Hence, we need to check if the probability mass exceeds the bound only at s_I. From here on, we only consider DTMCs with unique initial states.*

Given a DTMC $\mathcal{M} = (S,P,s_I,AP,L)$, PCTL Model Checking involves building the *satisfiability set* $Sat(\Phi)$ for the property Φ in consideration and check if $s_I \subseteq Sat(\Phi)$. In order to compute the satisfaction set for Φ, we need to compute the satisfaction sets for its sub-formulae. This leads to a recursive computation of the satisfaction sets. It translates to a bottom-up traversal of the parse tree of Φ. The parse tree of a PCTL formula has either a tt literal or an atomic proposition $a \in AP$ as a leaf node. Every inner node is an operator defined in PCTL grammar such as $\neg, \wedge, \bigcirc, \mathcal{U}, P_J$. At any level, the satisfaction set is built based on the satisfaction sets of the sub-formulae or the child nodes. Assuming Ψ is a sub-formula, after $Sat(\Psi)$ is calculated, a new atomic proposition a_Ψ is introduced. If a state $s \subseteq Sat(\Psi)$, then a_Ψ is added to $L(s)$. This information is used by the parent node to the calculate satisfaction sets. At the end of the traversal, the main formula Φ would be reduced to a_Φ.

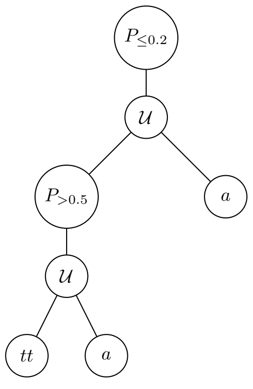
Example 2.3.2. *Let us consider a nested PCTL formula $P_{\leq 0.2}[P_{>0.5}[tt\mathcal{U}a]\mathcal{U}a]$. Let us denote this nested PCTL formula by Φ. Here, a is an atomic formula. Φ can be broken down into sub-formulae as shown in the tree representation in Figure 2.2(a). Each node is a sub-formula. The sub-formulae have been spelled out in* (2.2)*. Given $\mathcal{M}_1$ in Figure 2.1, we can do model checking with respect to Φ in order to check if $\mathcal{M}_1 \models \Phi$. Here, φ' and φ'' are path formulae. Let us consider $\varphi' = tt\mathcal{U}a$. From Figure 2.1, we can observe that $Sat(\varphi') = \{s_1, s_2, s_3, s_5, s_4\}$ since these are the 5 states on the relevant path.*

Moving up the parse three, we need to calculate the probabilities for the satisfaction sets of Ψ' since we need to know which states satisfy the bound $P_{>0.5}$. From the Table 2.1, we can say that among the states in $Sat(\varphi')$, which is the child node of Ψ', s_5 does not satisfy this bound. Hence, $Sat(\Psi') = \{s_1, s_2, s_3, s_4\}$.

We now move a level higher in the parse tree. Let us consider the path formula $\varphi'' = \Psi'\mathcal{U}a$. This requires a constrained reachability from Ψ' to an a-state. From the Figure 2.1, we can say that all the states in $Sat(\Psi')$ are reachable to the a-state, i.e., s_4. Hence, all the states that are part of $Sat(\Psi')$ satisfy the path formula φ''. Therefore, $Sat(\varphi'') = \{s_1, s_2, s_3, s_4\}$.

Moving up a level, we have now reached the root node of the parse tree. Since Φ is a P formula, we need to calculate the probabilities of all the states satisfying φ'' to check if the bound is satisfied. The probabilities $Pr(s \models \varphi'')$ are listed in the Table 2.2(b) which shows that all states violate $\Phi = P_{\leq 0.2}(\varphi'')$ including s_1 which is the initial state. Therefore, $\mathcal{M}_1 \not\models \Phi$.

$$\overbrace{P_{\leq 0.2}[\overbrace{\underbrace{P_{>0.5}[\underbrace{tt\mathcal{U}a}_{\varphi'}]}_{\Psi'}\mathcal{U}a}^{\varphi''}]}^{\Phi} \tag{2.2}$$



(a) Tree representation

s_i	$Pr_{\mathcal{M}_1}(s \models \varphi'')$
s_1	0.5
s_2	0.5
s_3	0.5
s_4	1
s_5	0
s_6	0

(b) Probabilities

Figure 2.2: PCTL formula $\Phi = P_{\leq 0.2}[P_{>0.5}[tt\mathcal{U}a]\mathcal{U}a]$

2.4 Properties of DTMCs

Properties are desirable behaviour for a DTMC. We list a set of properties we are interested in model checking. This section helps us identify the subset of properties such as *linear-time properties* and *simple reachability properties* that have already been addressed in [WJÁ+12b, WJÁ+12a] and distinguish them from the *full PCTL properties* addressed for the first time in this thesis.

Definition 2.4.1 (Linear-Time Property). *A linear-time (LT) Property over a set of atomic propositions (AP) defines a set of permissible traces $\gamma_0\gamma_1\gamma_2...$, with $\gamma_i \subseteq 2^{AP}$, essentially specifying the admissible behaviour of a system [BK08, WJÁ+12b].*

An LT Property is a subset of $(2^{AP})^\omega$ where $(2^{AP})^\omega$ is a set of infinite words over AP. Since we do not have accepting states but only absorbing states within a DTMC, it would be sufficient to define LT property in terms of infinite words.

Definition 2.4.2 (Reachability Property). *A reachability property $\Diamond\Phi$, where Φ is a PCTL state formula, is a property with a set of infinite traces that at some point satisfies the formula Φ.*

$$\Diamond\Phi = \{\gamma_0\gamma_1\gamma_2\ldots \subseteq (2^{AP})^\omega \mid \exists i \geq 0 : \gamma_i \models \Phi\} \tag{2.3}$$

Definition 2.4.3 (Simple Reachability Property). *Simple reachability property is a subclass of reachability property of the form $\Diamond a$ where $a \in AP$.*

$$\Diamond a = \{\gamma_0\gamma_1\gamma_2\ldots \subseteq (AP)^\omega \mid \exists i \geq 0 : a \in \gamma_i\} \tag{2.4}$$

A simple reachability property is a reachability property over an atomic formula.

Definition 2.4.4 (Constrained Reachability Property)**.** *A constrained reachability property $\Phi_a \mathcal{U} \Phi_b$, where Φ_a and Φ_b are PCTL formulae, is a property with a set of infinite traces such that Φ_a holds up until some point where Φ_b holds.*

$$\Phi_a \mathcal{U} \Phi_b = \{\gamma_0\gamma_1 \ldots \subseteq (2^{AP})^\omega | \, \exists n \geq 0 : \gamma_n \models \Phi_b \wedge (\forall\, 0 \leq i < n : \gamma_i \models \Phi_a)\} \tag{2.5}$$

If $n = 0$, then $\Phi_b \models \gamma_0$. In such a case, the property $\Phi_a \mathcal{U} \Phi_b$ is satisfied at the very first instance.

Definition 2.4.5 (Invariant)**.** *An invariant P_{inv} is an LT property with a set of infinite traces where a certain formula Φ always holds.*

$$P_{inv} = \{\gamma_0\gamma_1 \ldots \subseteq (2^{AP})^\omega | \, \forall i : \gamma_i \models \Phi\} \tag{2.6}$$

Remark (Invariants as reachable properties)**.** *An invariant needs to hold in every state of the reachable fragment within a DTMC. There should be no state, violating the invariant, reachable from any of the initial states. Hence, an invariant property can be transformed into a reachability property.*

Consider $P_{inv} = \Box\Phi$ which means Φ must always hold. Given the duality of $\Box$ operator, it can be written as $\neg\Diamond\neg\Phi$ [BK08, p.248].

if an invariant is wrapped with P operator, $P_{=1}[\Box\Phi]$, it is a PCTL formula. This means $s \models P_{=1}[\Box\Phi]$ if and only if every path from state s would always be a path where Φ holds. In other words, the probability of reaching a state where $\neg\Phi$ holds is 0 [BK08, p.789]. This means any invariant can be checked by doing a reachability analysis.

$$s \models P_{=1}[\Box\Phi] \;\equiv\; P_{=0}[\neg\Diamond\neg\Phi] \;\equiv\; \neg P_{>0}[\neg\Diamond\neg\Phi] \tag{2.7}$$

2.5 Probabilistic Counterexamples

One of the most important features of model checking involves generating counterexamples. Counterexamples form an invaluable piece of information for debugging. Generating counterexamples for probabilistic systems requires calculating the set of paths that collectively have a probability mass that violates the property. There are several approaches for doing this. [HKB09] explores the option of path-based counterexamples. It calculates the smallest counterexample, i.e., a counterexample that deviates the most from the required probability bound given that it has the smallest number of paths seen as a k shortest path problem. Due to the large number of paths that may be needed with the worst case being doubly exponential in problem size, they introduce regular expression as a succinct form of representing counterexamples. The other method is to have a *critical subsystem* which is a sub-DTMC that violates the property. This is always bound by the number of states of the DTMC in consideration. [WJÁ+12b, WJÁ+12a] build a linear constraint system that violates the property. It translates the counterexample generation into a state-minimization problem. It uses SMT and MILP solvers to compute the *minimal critical system.* We introduce some of these concepts regarding counterexamples here.

Critical Subsystem Critical Subsystems are one of the methods of providing evidence of violation of a property. It comprises a set of states reachable from the initial state that contribute to the violation of the property. It also contains the transitions between the states, making the evidence succinct as opposed to path-based counter examples.

Definition 2.5.1 (Critical Subsystem). *A critical subsystem of DTMC $\mathcal{M}$ for a PCTL property $P_{\leq\lambda}(\varphi)$ is a DTMC $\mathcal{M}' = (S',P',s_I^{'},L')$ such that $S' \subseteq S$, $s_I' = s_I$, $L'(s) = L(s)$,$P'(s,s') = P(s,s')$ for all $s,s' \in S'$ and $Pr^{s_I^{'}}_{\mathcal{M}'}(\varphi) > \lambda$ [WJÁ+12b].*

Remark. *The probability measure of the path formula φ at the initial state $s_I^{'}$ should violate the bound specified in the property. For the property $P_{>\lambda}[\varphi]$, the critical subsystem requires $Pr^{s_I^{'}}_{\mathcal{M}'}(\varphi) \leq \lambda$ to be satisfied*

Within a critical subsystem, every state is reachable from one of the initial states. The probability measure of the initial state with respect to the property in question needs to violate the bound. We are looking for the minimal possible number of states that act as evidence of violation. This leads us to define *state-minimal critical subsystems.*

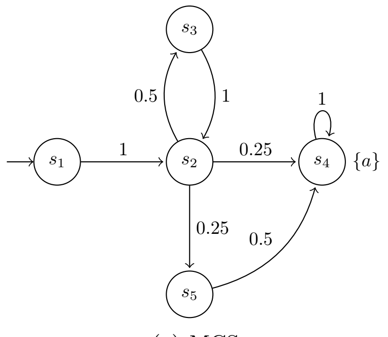
Definition 2.5.2 (Minimal Critical Subsystem). *A Minimal Critical Subsystem (MCS) is critical subsystem with the least number of states that violates a property.*

The focus in the thesis is state-minimality. Though the work can be extended to include transition-minimality by modifying constraints to minimize transitions instead of states while keeping the remaining conditions same.

Example 2.5.1. *In Example 2.3.2, through PCTL model checking, we have found that $\mathcal{M}_1$ (see Figure 2.1) violates $\Phi = P_{\leq 0.2}[P_{>0.5}[tt\mathcal{U}a]\,\mathcal{U}a]$, i.e., $\mathcal{M}_1 \not\models \Phi$. Let us compute the counter example in form of an MCS. Based on Table 2.3(b), where x_{s_i} is an inclusion variable to indicate if a state is a part of an MCS, we have an MCS with 5 states that is shown in Figure 2.3(a). It is the minimum critical subsystem because reduction of any more states would make the inner formula $P_{>0.5}[tt\mathcal{U}a]$ devoid of all states except s_4 in the satisfaction set since all these 5 states contribute to the probability measure.*

2.6 Probabilistic Model Checking

Model checking is a decision problem. The recursive computation of satisfiability sets for a formula can be translated to a set of constraints based on



(a) MCS

s_i	$Pr_{\mathcal{M}_1}(s \models \varphi'')$	x_{s_i}
s_1	0.5	1
s_2	0.5	1
s_3	0.5	1
s_4	1	1
s_5	0	1
s_6	0	0

(b) Probabilities

Figure 2.3: Minimal Critical Subsystem for $\Phi = P_{\leq 0.2}[P_{>0.5}[tt\mathcal{U}a]\mathcal{U}a]$

satisfaction relations for PCTL logic specified in 2.3.4. These set of constraints can be expressed in terms of *quantifier-free linear arithmetic.* We need to check satisfiability of these constraints and search for a feasible solution.

Definition 2.6.1 (Linear Arithmetic)**.** *The syntax of a formula in linear arithmetic is defined by the following rules:*

$$\begin{aligned} formula &: formula \land formula | (formula) | atom \\ atom &: sum\ op\ sum \\ op &: = | \leq | < \\ sum &: term \mid sum + term \\ term &: identifier \mid constant \mid constant\ identifier \end{aligned}$$

The operators $\geq$ and $>$ can be replaced by $\leq$ and $<$ if the coefficients are negated. We consider rational numbers and integers as domains. [KSB08, p.111].

Looking for MCS requires minimizing the set of states needed to violate a property subject to the a set of constraints. These constraints need to hold in order to satisfy the sub-formulae of the property in question. This can be seen as a minimization problem in optimization and can be framed as a *linear programming problem.*

Definition 2.6.2 (Linear Programming)**.** *A linear program in standard form, also called general form, has an objective function which needs to be minimized. The objective function is a linear function over a set of variables called decision variables which need to be assigned optimal values. A set of values for the variables is called a solution. A solution is feasible if it satisfies all constraints.*

$$\varsigma = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

where x_j for $1 \leq i \leq n$ are decision variables. ς is the objective function that *needs to be minimized and is subject to linear constraints of the form*

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 \ &+ \cdots + a_{1n}x \ &\bowtie b \\
a_{21}x_1 + a_{22}x_2 \ &+ \cdots + a_{2n}x \ &\bowtie b \\
\vdots \qquad\qquad &\vdots \\
a_{m1}x_1 + a_{m2}x_2 &+ \cdots + a_{mn}x_n &\bowtie b \\
x_1, x_2, \quad &\cdots \quad , x_n &\geq 0
\end{aligned}$$

where $\bowtie \in \{\leq\ ,\ <\ ,\ =\ ,\ >\ ,\ \geq\}$. $\bowtie$ is converted into an equality by adding a slack variable s where $s \bowtie b$ [KSB08, Van08]. It is called linear programming problem if the variables used are rational numbers. One of the common algorithms used for arithmetic over real variables is the *Simplex* algorithm. Other alternatives are the ellipsoid and interior point algorithms [DDM06]. Both these algorithms have been proved to have a polynomial time. Despite good practical performance of these methods, Simplex is the most popular method. Simplex algorithm has a worse-case exponential complexity but this is rarely encountered in practice. Analysis of the most likely problems using *shadow-vertex* simplex algorithm has shown a polynomial complexity [ST04].

2.6.1 Mixed Integer Linear Programming

If a linear program specified in 2.6.2 is defined over integer variables, it is called *integer programming.* If the variables are a mix of real-valued and integer variables, it is called *mixed integer linear program.* The common procedure used to solve such problems is called *branch and bound.*

Branch and bound procedure This involves initially relaxing the problem by dropping the integral constraints. The simplex algorithm is applied to this relaxed problem. If we get an integral solution, we terminate the procedure. Otherwise, we need to take the non-integral solution of the integer variable and round it off to nearest integers. If the the solution is for instance $x_1 = 1.6$, we continue the search by rounding it off to $x_1 < 1$ and using this as an additional bound. We also separately search with another bound $x_1 > 2$. Based on which branch gives the *best-so-far* optimal solution, we choose to enumerate the branch. If $x_1 > 2$ gives a lower value for objective function, we enumerate this branch. We continue building this *enumeration tree* and search for the optimal integral solution using a depth-first search. We prefer depth-first search to breadth-first search because we get integral solutions much deeper in the tree. If the problem is aborted before it terminates, we are more likely to have an integral solution. Additionally, enumerating a node results in adding additional bounds to the existing problem. A running simplex algorithm can modify the current dictionary to include the additional constraints. The assumption here is that the objective function is linear. Once we reach a node that yields an integral solution, we make the subsequent nodes leaf nodes since searching deeper generally yields sub-optimal solutions compared to the parent node. This process is called *pruning* that largely boosts the performance [Van08, pp.392-404].

If the variables have no upper or lower bounds, the branch and bound procedure may not terminate and may loop forever. Hence, we apply the *small-bound property* which essentially means having a small domain so that there are a set of finite values with which the variables can be instantiated [KSB08, DDM06].

2.6.2 SAT-Modulo Theories Solver

We can utilize *SAT Modulo Theories solver*, or in short an SMT solver to find a feasible solution for the linear constraint system. SMT solver comprises a *SAT solver* and a *Theory solver* (T-Solver). The SAT solver considers each constraint as a boolean variable and tries to satisfy the boolean skeleton of the *Conjunctive Normal Form* (CNF) formula given as the input. The T-Solver, based on the constraints by the SAT solver, checks for the consistency of constraints by assignment of variables. The SMT solver can yield a feasible solution if the problem is satisfiable (SAT). It will otherwise return unsatisfiable (UNSAT) if the decision procedure terminates.

The T-Solver can be instantiated with various theories. We instantiate it with quantifier free linear arithmetic (see 2.6.1). We use disjunctions here to define our constraints. One of the common algorithms used for arithmetic over real variables is *Simplex* algorithm.

A variant of Simplex algorithm is used in the SMT solver called *general Simplex*. It involves only the *phase 1* of simplex algorithm that deals with searching for the *basic feasible solution*, a solution that satisfies all the constraints. It does not involve *phase 2* dealing with an optimal solution, i.e., minimizing the objective function.

MILP problems are solved by SMT solvers by using a variant of branch and bound procedure that works in conjunction with the simplex algorithm. The SMT Solver looks only for a satisfiable solution. Hence, in order to find the MCS, we need to search for the optimal solution and this is done using Binary search that is explained later in 3.1.3.

2.6.3 MILP Solver

An MILP Solver generally uses branch and bound procedure in conjunction with simplex algorithm to search for an optimal solution similar to the description in 2.6.1. We feed an objective function that returns an optimal solution if a solution is available. We need not explicitly search for an optimal solution. The grammar used to define constraints is quantifier free linear arithmetic without disjunctions. The absence of disjunctions enormously speeds up the search.

2.7 MCS for Reachable Properties

The work until now in the area of computation of MCS through SMT and MILP solvers considers simple reachability properties of the form $P_{\leq\lambda}[\Diamond a]$ where a is an atomic formula [WJÁ+12a]. Let the DTMC in consideration be $\mathcal{M} = (S,P,\iota_{init},AP,L)$. Let $T_a = \{s \in S \,|\, a \in L(s)\}$ be the target states. For the SMT formulation, we have a characteristic variable $x_s \in [0,1] \subseteq \mathbb{R}$. $x_s = 0$ or $x_s = 1$ are the values allowed by the formulation. Let $p_s \in [0,1] \subseteq \mathbb{R}$ be the variable that is assigned with the reachability probabilities. Following is the SMT formulation.

$$\text{minimize} \sum_{s \in S} x_s \tag{2.8a}$$

such that

$$\forall s \in T_a : (x_s = 0 \wedge p_s = 0) \oplus (x_s = 1 \wedge p_s = 1) \tag{2.8b}$$

$$\forall s \in S \setminus T_a : (x_s = 0 \wedge p_s = 0) \oplus (x_s = 1 \wedge p_s = \sum_{s' \in succ(s)} P(s,s') \cdot p_{s'}) \tag{2.8c}$$

$$p_{s_I} > \lambda. \tag{2.8d}$$

The formulation assigns probability values based on the linear equation system mentioned in (2.1) and the number of states are minimized to compute an MCS. The MILP formulation allows for integer variables. Hence $x_s \in [0,1] \subseteq \mathbb{Z}$. Following is the MILP formulation.

$$\text{minimize} \; -\frac{1}{2}p_{s_I} + \sum_{s \in S} x_s \tag{2.9a}$$

such that

$$\forall s \in T_a : p_s = x_s \tag{2.9b}$$

$$\forall s \in S \setminus T_a : p_s \leq x_s \tag{2.9c}$$

$$\forall s \in S \setminus T_a : p_s \leq \sum_{s' \in succ(s)} P(s,s') \cdot p_{s'} \tag{2.9d}$$

$$p_{s_I} > \lambda. \tag{2.9e}$$

(2.9a) ensures that if a target state is a part of the critical subsystem, it is assigned a probability 1 else assigned 0. For the remaining states, (2.9c) assigns a probablity 0 to the states not included in the critical subsystem. (2.9d) assigns an upper bound on the probability. To get maximal probabilities, we need to maximize the probability of the initial state which is the reason it is a part of the objective function in (2.9a).

The computation MCS for PCTL properties is based on the same idea. However, we need to build constraints for the satisfiability of all the sub-formulae along with a similar set of constraints for the computation of MCS. This is further explained in the following chapter.

Chapter 3

MCS for PCTL Properties

In order to compute a minimal critical subsystem (MCS) for a property specified in PCTL, it is required to check whether the PCTL property is violated within a DTMC $\mathcal{M}$. To check whether the PCTL property holds within $\mathcal{M}$, we need to do PCTL model checking. This involves traversing through the parse tree of the PCTL formula and computing the set of states where each of the sub-formula holds. It also involves calculating probabilities for a state formula with P operator. These probabilities and satisfiability of states are used to compute the relevant path for the outer formula within the PCTL property. In case the probability assigned to the initial state violates the bound specified, the property is violated within $\mathcal{M}$.

Difference between MCS for reachability over atomic formula and PCTL formula A reachability property is a probability bound reachability to any state that satisfies a PCTL formula such as $P_{\leq\lambda}[\lozenge\Phi]$ where Φ is a PCTL formula. The previous work on computing MCS only considers the case of simple reachability properties (see 2.4) and only handle atomic propositions. The current work extends this approach to handle all PCTL formulae both in form of reachability properties and constrained reachability. This means we first need to have a set of constraints that define satisfiability for all the subformulae. We build the satisfaction sets traversing through the parse tree of PCTL formula. We can then check if the DTMC violates a property at the initial state. If violated, compute an MCS based on the satisfaction sets of all the subformulae. This means unlike in [WJÁ+12b], where the MCS has only the states on the relevant path based on reachability analysis, an MCS based on a nested PCTL formula may have states that are not necessarily on the relevant path of the outer until formula. However, these states may need to be a part of the subsystem since it contributes to the satisfiability of the sub-formulae.

Example 3.0.1. *In the Example 2.5.1, we see that though the state s_5 is not a part of the relevant path with respect to the outer until formula in 2.2, i.e., φ, it is needed since it contributes to the probabilities of all the states on the relevant* *path. Hence, a straightforward reachability analysis is null and void.*

Computation of MCS as a minimization problem The computation of MCS can be seen as a minimization problem. The major steps involved are *global model checking* that involves building satisfiability sets for the formulae and computation of a minimum set of states that violate the probability bound. We will need a few variables to do this. Since we are computing satisfiability of states with respect to each sub-formula ϕ, we can have a variable for a combination of a state and a formula, i.e., $\sigma_{(\phi,s)}$. We need a variable to store probabilities at every state with respect to a path formula φ, i.e., $p_{(\varphi,s)}$. We also need an inclusion variable that denotes whether a state is included within an MCS, i.e. x_s. This is the decision variable that is part of the objective function. We need to assign domains, impose upper and lower bounds to these variables. All the variables are real for a linear program while we can have combination of real and integer variables for MILP.

We impose linear constraints on $\sigma_{(\phi,s)}$ with respect to every formula based on the satisfaction relation as defined in 2.3.4. We impose linear constraints on $p_{(\phi,s)}$ for every path formula and a state formula that wraps this path formula. Traversing through the parse tree and repeating this process would result in a set of linear constraints.

PCTL properties would need to be satisfied with an upper or lower probability bound. In order to build an MCS, we add a constraint that ensures the initial state violates this probability bound. At this point, we have a set of constraints that can build a critical subsystem that violates the property. Thus we have converted the computation of MCS to a minimization problem.

3.1 SMT formulation

The SMT formulation has constraints over quantifier-free linear arithmetic. The formulation allows disjunctions over constraints. The variables are real valued though integral constraints are imposed on a few of them. Let us define the domain and intervals for the variables used.

Satisfiability Variable The characteristic variable $\sigma_{(\Phi,s)} \in [0,1] \subseteq \mathbb{R}$ is used to keep track of the satisfiability of a formula Φ in a state $s \in S$. The characteristic variable $\sigma_{(\Phi,s)}$ is assigned 1 if Φ is satisfied in state s. Otherwise is assigned 0. The satisfaction relation is based on 2.3.4.

$$\sigma_{(\Phi,s)} = \begin{cases} 1 & \text{if } \Phi \models s \\ 0 & \text{if } \Phi \not\models s \end{cases}$$

Since $\sigma_{(\Phi,s)}$ is a real-valued variable, we need to impose an integral constraint on it. We use XOR in order to achieve this in the following constraint.

$$\sigma_{(\phi,s)} = 0 \oplus \sigma_{(\phi,s)} = 1 \tag{3.1}$$

State Inclusion Variable The state inclusion variable $x_s \in [0,1] \subseteq \mathbb{R}$ is the decision variable. It is assigned 1 if the state $s \in S$ is a part of the MCS C. Otherwise is assigned 0.

$$x_s = \begin{cases} 1 & \text{if } s \in C \\ 0 & \text{if } s \notin C \end{cases}$$

Following is the integral constraint.

$$x_s = 0 \oplus x_s = 1 \tag{3.2}$$

Probability Measure In order to assign probability values used to build satisfiability sets for formula $P_{\leq\lambda}[\varphi]$, we have $p_{(\varphi,s)} \in [0,1] \subseteq \mathbb{R}$. This variable holds the probability measure $Pr_{\mathcal{M}}(s \models \varphi)$ of a path formula φ at state $s \in S$ in a DTMC $\mathcal{M} = (S,P,s_I,AP,L)$.

3.1.1 State Formulae

The state formulae is valid for a single state. A state formula can be an atomic formula or a conjunction of two PCTL formulae, a negation of a PCTL formula or a formula with P operator with a probability bound wrapping a path formula. Let us consider the individual cases and define the satisfiability constraints.

True literal Let tt be the true literal. The characteristic variable $\sigma_{(tt,s)}$ is assigned 1, regardless of the label at the state s, as long as it is part of the critical subsystem.

$$\boxed{tt}$$

$$\sigma_{(tt,s)} \leftrightarrow x_s \tag{3.3}$$

Atomic formula Let $\Phi \equiv a$ be an atomic formula where $a \in AP$. $\sigma_{(\Phi,s)}$ is a characteristic variable for a state $s \in S$. The satisfaction relation as mentioned in 2.3.4 says $s \models a$ iff $a \in L(s)$. If the atomic proposition a is contained in the label of the state s, then the atomic formula $\Phi \equiv a$ is satisfied. However, our goal is to compute an MCS and any state outside the MCS is of no interest to us. Such a state should not contribute to the satisfiability of other states within an MCS. Hence, we impose a constraint that ensures Φ will not hold in a state s if it is not a part of the MCS, regardless of its label.

$$\boxed{\Phi \equiv a}$$

$$\sigma_{(\Phi,s)} = \begin{cases} 0 & \text{if } L(s) \neq a \\ x_s & \text{if } L(s) = a \end{cases} \tag{3.4}$$

If the $L(s) \neq a$, $\sigma_{(\Phi,s)}$ is always 0. If $L(s) = a$, $\sigma_{(\Phi,s)}$ is assigned 1 if and only if the state is included in the critical system.

Negation of a formula If we have a formula of the form $\Phi \equiv \neg\Phi'$, where Φ' is another PCTL formula, we have to assign values to $\sigma_{(\Phi,s)}$ based on the sub-formula $\sigma_{(\Phi',s)}$. Hence, we need to recursively compute $\sigma_{(\Phi,s)}$. The satisfaction relation is $s \models \neg\Phi$ iff $s \not\models \Phi$. This means $\sigma_{(\Phi,s)}$ needs to be assigned with the complement of $\sigma_{(\Phi',s)}$. However, it will be assigned 1 if and only if s is part of the critical subsystem.

$$\boxed{\Phi \equiv \neg\Phi'}$$

$$\sigma_{(\Phi,s)} = 1 \leftrightarrow \sigma_{(\Phi',s)} = 0 \wedge x_s = 1 \tag{3.5}$$

Conjunctions A formula of the form $\Phi \equiv \Phi_1 \wedge \Phi_2$ is a conjunction of two sub-formulae. The satisfaction relation says $s \models \Phi_1 \wedge \Phi_b$ iff $s \models \Phi_1$ and $s \models \Phi_b$. This means that both Φ_1 and Φ_b need to hold simultaneously at state s for $\Phi_1 \wedge \Phi_b$ to hold. It is sufficient to decide the value of $\sigma_{(\Phi_1\wedge\Phi_2,s)}$ with respect to the sub-formulae without taking into account the inclusion variable x_s because we assume that the inclusion variable has been factored in within the sub-formulae. Hence, the sub-formulae would not hold if s is not a part of the critical subsystem.

$$\boxed{\Phi \equiv \Phi_1 \wedge \Phi_2}$$

$$\sigma_{(\Phi_1\wedge\Phi_2,s)} = 1 \leftrightarrow \sigma_{(\Phi_1,s)} = 1 \wedge \sigma_{(\Phi_2,s)} = 1 \tag{3.6}$$

P formula A P formula is of the form $P_{\bowtie\lambda}[\varphi]$ where $\lambda \in [0{,}1] \subseteq \mathbb{R}$ and $\bowtie$ could refer to strict or non-strict bounds. φ here is a PCTL path formula. Since we need to discuss path formulae, it is premature to introduce it here. P formula will be juxtaposed with path formulae.

3.1.2 Path Formulae

A path formula is defined for a path. We say a characteristic variable $\sigma_{(\varphi,s)}$ for a path formula φ that holds in a state s if the state is on a path that satisfies φ and all the states in the path must be a part of the critical subsystem.

Next Path Formula A next formula is of the form $\varphi \equiv \bigcirc\Phi$. It is also denoted by $\mathcal{X}\phi$ where the sub-formula Φ needs to be a state formula as specified in 2.3.2. Let $\pi = s_0 s_1 s_2 ...$ be a path. The satisfaction relation says that $\pi \models \bigcirc\Phi$ iff $\pi[1] \models \Phi$. That is $\sigma_{(\bigcirc\Phi,s)}$ is assigned 1 if there exists at least one successor state $s' \in succ(s)$ such that $s' \models \Phi$. Here, $succ(s)$ is a *successor function.* We also need to make sure that state s is included within the critical subsystem, along with its successor s' where Φ holds. We enforce that s is part of the critical subsystem. However, since the satisfiability variable $\sigma_{(\Phi,s')}$ factors in the inclusion variable for s', we do not explicitly enforce it here.

$$\boxed{\varphi \equiv \bigcirc\Phi}$$

$$\sigma_{(\bigcirc\Phi,s)} = 1 \leftrightarrow x_s = 1 \wedge \bigvee_{s' \in succ(s)} \sigma_{(\Phi,s')} = 1 \tag{3.7}$$

Until Path Formula An until path formula is denoted by $\varphi \equiv \Phi_a \,\mathcal{U}\, \Phi_b$. The satisfaction relation says $\pi \models \Phi_a \,\mathcal{U}\, \Phi_b$ iff $\exists j \geq 0 : \ \pi[j] \models \Phi_b \wedge (\forall 0 \leq i < j : \pi[i] \models \Phi_1)$. Satisfying an until path formula is a constrained reachability problem. The formula $\Phi_a \,\mathcal{U}\, \Phi_b$ holds for a certain path π, if every state s_i encountered in the path satisfies Φ_a until we reach a state s_j that satisfies Φ_b where $i < j$. If a state s lies on such a path π and all the states in this path are a part of the critical subsystem, then $\sigma_{(\varphi,s)}$ is assigned 1. This can be expressed through the following constraints.

$$\boxed{\varphi \equiv \Phi_a \,\mathcal{U}\, \Phi_b}$$

$$\sigma_{(\Phi_a \mathcal{U} \Phi_b,s)} = 1 \leftrightarrow \sigma_{(\Phi_b,s)} = 1 \vee (\sigma_{(\Phi_a,s)} = 1 \wedge \bigvee_{s' \in succ(s)} \sigma_{(\Phi_a \mathcal{U} \Phi_b,s')} = 1) \tag{3.8}$$

Remark. *If all $\neg\Phi_a$ state including all Φ_b-states in the DTMC are made into absorbing states, the constrained reachability problem is reduced to a reachability problem [BK08]. However, this is only possible if Φ_a and Φ_b are atomic formulae* *because of the dependence of the outer formulae on the inner formulae explained in 3.1.1 and 3.1.3.*

Example 3.1.1. *Consider the property $P_{\leq 0.8}[a\,\mathcal{U}\,P_{>0}[\mathcal{X}b]]$ and the DTMC in Figure 3.1. $s_1 \models a$ and $s_2 \models P_{>0}[\mathcal{X}b]$ because $s_3 \models b$. Now if s_1 and s_2 are made absorbing states, within such a system $s_2 \not\models P_{>0}[\mathcal{X}b]$ since s_3 is not reachable. Due to the dependence of a formula on its sub-formulae, this method of transforming constrained reachability to reachability may not work unless Φ_a and Φ_b are atomic formulae in the until formula $\Phi_a\,\mathcal{U}\,\Phi_b$*

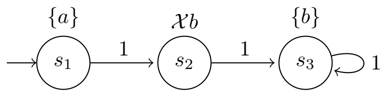


Figure 3.1: DTMC $\mathcal{M}$ checked for the property $P_{\leq 0.8}[a\,\mathcal{U}\,P_{>0}[\mathcal{X}b]]$

Example 3.1.2. *One of the inadequacies of* (3.8) *is handling of a self-loop or a bigger loop. Let us consider the DTMC in Figure 3.2 that has a loop. $\hat{\pi} = s_1s_2s_3s_1$ is the path fragment with the loop. Every state in $\hat{\pi}$ is a Φ_a-state. Let be ν an assignment function. Let us assume that the solver does the following assignment. $\nu(\sigma_{(\varphi,s_1)}) = 1$ and $\nu(\sigma_{(\varphi,s_2)}) = 1$. Now s_3 has a successor s_1 that has already been assigned* 1 *for the path formula. Having assigned* 1 *to $\sigma_{(\varphi,s_3)}$,* (3.8) *is locally satisfied without including Φ_b-state. This is because* (3.8) *is a disjunction of two constraints and only the second constraint is satisfied. Hence an additional constraint needs to be added to in order to ensure every such path ends with a Φ_b-state.*

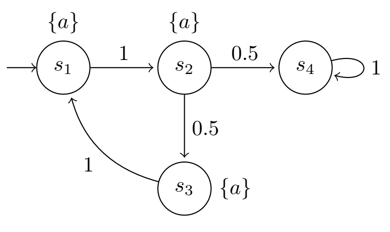


Figure 3.2: DTMC $\mathcal{M}$ with a loop

Backward reachability for inner until formula Let $\varphi \equiv \Phi_a\,\mathcal{U}\,\Phi_b$. Backward reachability ensures that in the case $\sigma_{(\varphi,s)}$ is assigned 1, it is done so if and only if it is on a path that ends with Φ_b-state. We do this by introducing a new variable $r_s^{\rightarrow} \in [0,1] \subseteq \mathbb{R}$. This variable is introduced to handle loops in a path. $r_s^{\rightarrow} < r_{s'}^{\rightarrow}$ ensures that an assignment does not terminate at a loop since the successor state $r_{s'}^{\rightarrow}$ needs to be assigned with a greater value than its predecessor $r_s^{\rightarrow}$. In Example 3.1.2, the situation where the solver looks for a successor of s_3 now excludes s_1 as a candidate since there is a partial order introduced. The constraint ensures that if $\sigma_{(\varphi,s)}$ has been assigned with 1, then a successor of state s which either is a Φ_a-state or a Φ_b-state is also assigned maintaining a partial order. However, if s is a Φ_b-state, the constraint is satisfied and the solver does not assign the successors of such a Φ_b-state.

$$\boxed{\varphi \equiv \Phi_a \,\mathcal{U}\, \Phi_b}$$

$$\sigma_{(\varphi,s)} = 0 \vee \sigma_{(\Phi_b,s)} = 1 \vee \bigvee_{s' \in succ(s)} (\sigma_{(\varphi,s')} = 1 \wedge r_s^{\rightarrow} < r_{s'}^{\rightarrow}) \tag{3.9}$$

Bounded Until Path Formula A bounded until path formula is denoted as $\varphi \equiv \Phi_a \mathcal{U}^{\leq n} \Phi_b$. The satisfaction relation is

$$\pi \models \Phi_a \mathcal{U}^{\leq n} \Phi_b \text{ iff } \exists 0 \leq j \leq n : \pi[j] \models \Phi_b \wedge (\forall 0 \leq i < j : \pi[i] \models \Phi_a)$$

The bounded until is similar to the until formula with an additional condition. It needs to be satisfied within $n \in \mathbb{N}$ transitions. That means a path π satisfies bounded until formula $\Phi_a \mathcal{U}^{\leq n} \Phi_b$ if the Φ_b-state is reached within $n \in \mathbb{N}$ transitions going only through Φ_a-states. The constraint is a recursion with each successive recursion having one less transition within which Φ_b has to be satisfied.

$$\boxed{\varphi \equiv \Phi_a \mathcal{U}^{\leq n} \Phi_b}$$

$$\sigma_{(\Phi_a \mathcal{U}^{\leq n} \Phi_b,s)} = 1 \leftrightarrow \sigma_{(\Phi_b,s)} = 1 \vee (\sigma_{(\Phi_a,s)} = 1 \wedge \bigvee_{s' \in succ(s)} \sigma_{(\Phi_a \mathcal{U}^{\leq n-1} \Phi_b,s)} = 1) \tag{3.10}$$

Here, $n \geq 0$ and

$$\sigma_{(\Phi_a \mathcal{U}^{=0} \Phi_b,s)} = 1 \leftrightarrow \sigma_{(\Phi_b,s)} = 1 \tag{3.11}$$

If the number of transitions left is 0, then evidently the current state needs to satisfy Φ_b for the bounded until formula to be satisfied.

P Formula We return to a P formula having visited various path formulae it encapsulates. A P formula is of the form $\Phi \equiv P_{\bowtie\lambda}(\varphi)$. Let $p_{(\varphi,s)}$ be a variable to assign probabilities. $p_{(\varphi,s)}$ needs to be assigned with the sum of the probabilities of all paths π starting from a state s that satisfy the path formula φ within the critical subsystem.

$$p_{(\varphi,s)} = Pr_s^{\mathcal{M}}(s \models \varphi)$$

$$\sigma_{(P_{\bowtie\lambda}(\varphi),s)} = 1 \leftrightarrow p_{(\varphi,s)} \bowtie \lambda \tag{3.12}$$

Here, $\bowtie = \{< , \leq , = , \geq , >\}$. If the probability value is within the specified bound, the satisfiability variable $\sigma_{(P_{\bowtie\lambda}(\varphi),s)}$ is assigned 1.

Probability Measure for Next Path Formula The probability variable $p_{(\varphi,s)}$ for the next path formula $\varphi \equiv \bigcirc\Phi$ is subjected to the following constraints.

$$\boxed{\varphi \equiv \bigcirc\Phi}$$

$$\sigma_{(\varphi,s)} = 0 \leftrightarrow p_{(\varphi,s)} = 0 \tag{3.13a}$$

$$\sigma_{(\varphi,s)} = 1 \rightarrow p_{(\varphi,s)} = \sum_{s' \in succ(s)} P(s,s') \cdot \sigma_{(\Phi,s')} \tag{3.13b}$$

(3.13a) ensures that if a state s is not on a path that satisfies $\bigcirc\Phi$, then $p_{(\varphi,s)}$ is assigned 0. (3.13b) says that if a state s is on the path satisfying $\bigcirc\Phi$, then its probability value is $P(s,s') \cdot \sigma_{(\Phi,s')}$. This means that, if the successor is a Φ-state, then the probability assigned is the transition probability to the Φ-state. If the state s has more than one such successor, then the probability is the summation of the transition probabilities from state s to the Φ-states.

Probability Measure for Until Formula The probability variable $p_{(\varphi,s)}$ for until path formula $\varphi \equiv \Phi_a \mathcal{U} \Phi_b$ is subjected to following constraints.

$$\boxed{\varphi \equiv \Phi_a \mathcal{U} \Phi_b}$$

$$\sigma_{(\Phi_b,s)} = 1 \rightarrow p_{(\varphi,s)} = 1 \tag{3.14a}$$

$$\sigma_{(\varphi,s)} = 0 \leftrightarrow p_{(\varphi,s)} = 0 \tag{3.14b}$$

$$\sigma_{(\Phi_a,s)} = 1 \wedge \sigma_{(\Phi_b,s)} = 0 \rightarrow p_{(\varphi,s)} = \sum_{s' \in succ(s)} P(s,s') \cdot p_{(\varphi,s')} \tag{3.14c}$$

(3.14a) ensures that if we have a Φ_b-state, this implies that the probability that φ holds in such a state is 1. (3.14b) ensures that if a state is not on a path that satisfies φ, then the probability assigned is 0. (3.14c) assigns probabilities for Φ_a-states. A state can have multiple labels. If a state has both Φ_a and Φ_b, we make sure that (3.14c) ignores this and (3.14a) instead assigns the probability to it treating it as a Φ_b-state. This is to avoid a conflict between (3.14a) and (3.14c). (3.14c) assigns probabilities to Φ_a-states based on linear equation system mentioned in (2.1).

3.1.3 Minimal Critical Subsystem

Given a property of form $P_{\leq}[\varphi]$, we need to compute a minimal critical subsystem that violates it. Which means the initial state s_I needs to violate the bound.

$$p_{(\varphi,s_I)} > \lambda \tag{3.15}$$

In order for this bound to be violated, we need a set of states that contribute to the probability of the path formula φ. This is achieved by adding constraints for the satisfiability of φ. If it is a nested formula, the constraints are added recursively. The objective is to get a solution with minimum possible states that violate the bound. Hence, inclusion variables x_s are the *decision variables* part of the objective function. Computation of an MCS can be summarized with the following constraints.

$$\begin{aligned} &\text{minimize} \sum_{s \in S} x_s \\ &\text{such that} \\ &\qquad p_{(\varphi,s_I)} > \lambda \end{aligned}$$

Since SMT solvers cannot be used to minimize an objective function, we use the method of a binary search to look for an MCS.

Binary Search We start with the number of states in the DTMC as the upper bound (u) for the subsystem and 1 as the lower bound (l). This is the interval to be searched. The midpoint (m) is calculated. This midpoint is used build a constraint with an upper bound for the set of states in the critical subsystem.

$$\sum_{s \in S} x_s \leq m \tag{3.17}$$

A backtracking point is created before adding this to the set of existing constraints. The set of constraints can be seen as a stack and (3.17) is pushed to the top of this stack. We check the feasibility of this system. If the solver calculates a solution returning *SAT*, this midpoint m is set as the upper bound u of the new interval to be searched. We search a new midpoint and impose a tighter bound for the critical subsystem.

If the solver returns *UNSAT*, this means we cannot build a critical subsystem with utmost m states. We pop this constraint and return to the backtracking point that was created. The midpoint m becomes the lower bound l and the upper bound remains the same. A new midpoint is calculated and we search for a critical subsystem with a relaxed bound.

We repeat these steps up until a point where the lower bound is greater than the upper bound. When we exit this loop, we return the last satisfiable solution as the minimal critical subsystem.

3.1.4 Optimizations

We add redundant constraints that decrease the search space for the solver. Though these constraints do no exclude any solutions, they only make the possibility of assignments to get a satisfiable solution smaller.

Bound Cuts from Binary Search This work introduces *bound cuts* composed of upper and lower bounds on the number of states in the critical subsystem. During the binary search, we look for a solution with the midpoint of the first interval. After the solver returns the result, based on whether it is *SAT* or *UNSAT*, we set the midpoint as the upper or lower bound of the new interval respectively. The work preceding this calculates the new midpoint, pops the old upper bound and adds the new midpoint as the upper bound. The problem with this method is the solver loses any information of the feasible or infeasible bounds. For instance, if there are 100 states in the DTMC, the first upper bound is the midpoint of this interval of $[1,100] \subseteq \mathbb{Z}$, i.e., $\sum_{s\in S} x_s \leq 50$. If there is no solution, we pop the upper bound of 50 and search with 75 as the upper bound. We have lost the information that we cannot build a subsystem with utmost 50 states.

After popping the constraint, $\sum_{s\in S} x_s \leq 50$, bound cuts optimization adds this as a lower bound, i.e., $\sum_{s\in S} x_s > 50$. In case the solver returns *SAT* for an interval of $(50,75] \subseteq \mathbb{Z}$, we retain the bound $\sum_{s\in S} x_s \leq 75$. This acts as a redundant constraint to a tighter bound that follows, i.e., $\sum_{s\in S} x_s \leq 63$. Every time *UNSAT* is returned, the search space is drastically reduced and every time *SAT* is returned, a redundant constraint constraint is added.

Forward Cut Forward Cut ensures if a state s is included in a critical subsystem, i.e.,x_s is assigned 1, there exists a state s', a predecessor of s, that is also included in the critical subsystem. This constraint applies to all the states excluding the initial state s_I. This can be ensured with the following constraint.

$$\forall s \in S \setminus \{s_I\} : \; x_s = 1 \rightarrow \bigvee_{s' \in pred(s)\setminus\{s\}} x_{s'} = 1 \tag{3.18}$$

Satisfiability Cut Satisfiability cut ensures that a state s is part of the critical subsystem, i.e., x_s is assigned 1, if and only if there is at least one sub-formula within the property in consideration that holds in the state.

$$\forall s \in S \setminus \{s_I\} : \; x_s = 1 \leftrightarrow \bigvee_{i=0}^{n} \sigma_{(\psi_i, s)} = 1 \tag{3.19}$$

where n is the number of sub-formulae in the property and ψ is either a path or a state sub-formula.

Forward Reachability Forward Reachability ensures if a state s is included in a critical system, denoted by $\nu(x_s) = 1$, there exists a path from a state s to the initial state $s_I \subseteq S$. This can be ensured with the following constraint.

$$\forall s \in S \setminus \{s_I\} : \; (x_s = 0 \vee \bigvee_{s' \in pred(s)} x_{s'} = 1 \wedge r_s^{\leftarrow} < r_{s'}^{\leftarrow}) \tag{3.20}$$

The constraint ensures that if a state $\nu(x_s) = 1$, it needs to have a predecessor that is also included in the critical system and this predecessor needs to have another predecessor. There is a continuing path that this constraint establishes. The path terminates at an initial state s_I. A variable $r_s^{\leftarrow} \in [0{,}1] \subseteq \mathcal{R}$ is used to assign a value to every state s_i defining a partial order such that $r_{s_i}^{\leftarrow} < r_{s_{i+1}}^{\leftarrow}$ where $s_{i+1} = pred(s_i)$ that ensures the assignments do not stop at a loop.

Backward Reachability Backward Reachability ensures if a state s is included in a critical system, denoted by $\nu(x_s) = 1$, there exists a path from a state s to one of the final states $s_F \subseteq S$ where $s_F = \{s \in S \mid s \models \Phi_b\}$. This can be ensured with the following constraint.

$$x_s = 0 \vee \sigma_{(\Phi_b, s)} = 1 \vee \bigvee_{s' \in succ(s)} (\sigma_{(\varphi, s')} = 1 \wedge x_{s'} = 1 \wedge r_s^{\rightarrow} < r_{s'}^{\rightarrow})$$

This is similar to forward reachability. The constraint, however, ensures that if a state $\nu(x_s) = 1$, it needs to have a successor that is also included in the critical system. There is a continuing path that this constraint establishes. The path terminates at one of the final states s_F. This constraint also ensures that every successor state s' satisfies the formula φ or Φ_b. A variable $r_s^{\rightarrow} \in [0{,}1] \subseteq \mathcal{R}$ is used to assign a value to every state s_i defining a partial order such that $r_{s_i}^{\rightarrow} < r_{s_{i+1}}^{\rightarrow}$ where $s_{i+1} = succ(s_i)$. This partial ordering of states ensures the assignments do not terminate with a loop. Backward Reachability can only be used for until formulae of the form $\Phi_a \mathcal{U} \Phi_b$ where Φ_a and Φ_b are atomic formulae. It cannot be used for nested PCTL formulae.

Example 3.1.3. *This example demonstrates how to compute an MCS and why backward reachability cannot be used for nested PCTL formulae. Consider the DTMC $\mathcal{M}_2$ in Figure 3.3(a). We are checking if $\Phi = P_{<0.1}[P_{\geq 0.3}[tt\mathcal{U}s]\mathcal{U}t]$ is violated by $\mathcal{M}_2$.* (3.21) *identifies the sub-formulae within Φ.*

$$\overbrace{P_{<0.1}[\overbrace{\underbrace{P_{\geq 0.3}[\underbrace{tt\mathcal{U}s}_{\varphi'}]}_{\Phi'}\ \mathcal{U}t}^{\varphi}]}^{\Phi} \tag{3.21}$$

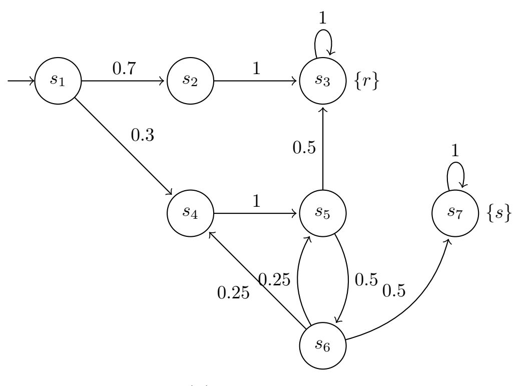
Let us first consider the innermost until formula, i.e., $\varphi' = tt\mathcal{U}s$*. We can observe in the Figure 3.3(a) the states that are on the relevant path of* φ'*, i.e.,* $S^{rel}_{\mathcal{M}_2} = \{s_1,s_4,s_5,s_6,s_3\}$*. Here* s_3 *is the target state because* $L(s_3) = s$*. Now,* $p_{(\varphi',s_3)}$ *can be assigned* 1 *based on* (3.14a)*. If the state is not on the relevant path of* φ'*, then probabilities of such states is 0* (3.14b)*. We use the linear equation system specified in* (3.14c) *to calculate the probabilities for the remaining states on the relevant path. For instance,* $p_{(\varphi',s_1)} = P(s_1,s_4) \cdot p_{(\varphi',s_4)}$*. Solving these equations for the remaining states would yield the probabilities listed in Table 3.3(b). Given all the probabilities are greater than* 0.3*, all the states on the relevant path are satisfiable with respect to* Φ'*.*

We have to find the relevant states for φ *given* $Sat(\Phi') = \{s_1,s_4,s_5,s_6,s_3\}$ *and* $Sat(t) = s_7$*. Finding the relevant states for* φ *is a problem of constrained reachability. Hence, we need to find the paths leading to* s_7 *through the set of states in* $Sat(\Phi')$*. We can see in Figure 3.3(a) that all states in* $Sat(\Phi')$ *lead to* s_7 *except for* s_3*. Hence* $p_{(\varphi,s_3)}$ *along with other states, i.e.,* $S \setminus Sat(\Phi')$ *are assigned probability* 0*.* $p_{(\varphi,s_7)}$ *is assigned* 1*. Solving the linear system based on* (3.14c)*, we get the probabilities mentioned in Table 3.3(d).*

The initial state violates the bound < 0.1*. Therefore, the property is violated, i.e.,* $\mathcal{M}_2 \not\models \Phi$*. We need to build an MCS that violates this property. This means we need enough states from the relevant path of* φ *to violate the probability bound* < 0.1*. Figure 3.3(c) is the MCS for* Φ*. The MCS has* s_3 *which is not a part of the relevant path of* φ*. However, it is required to satisfy* φ*. This is because* s_3 *is required to satisfy* Φ' *required for* φ*. We can conclude from this example that we may require states that are not on the relevant path of the outer until formula to be a part of the MCS because of the dependence of the outer formula over the inner sub-formulae. Therefore, one cannot enforce every state to be reachable from the target or final state of the outer until formula.*

3.2 MILP Formulation

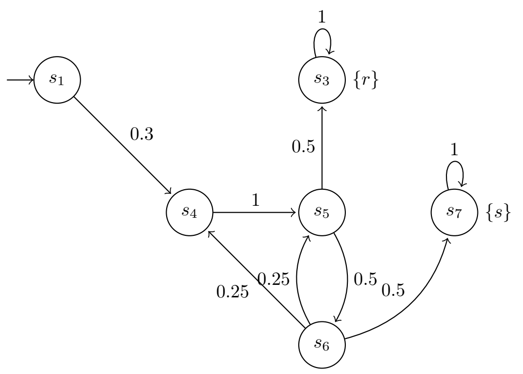
While the SMT formulation has disjunctive linear arithmetic constraints, mixed integer linear programming (MILP) formulation consists only of a *conjunction* of linear arithmetic constraints. These constraints have a mix of integer and real variables. The absence of disjunctions considerably speeds up the search for an optimal solution. Table 3.1 summarizes the variables used for the MILP formulation. It introduces new variables to handle transitions used for forward and backward reachability. It also defines an integer domain for a subset of variables without a need to explicitly specify integral constraints.



(a) DTMC $\mathcal{M}_2$

s_i	$p_{(\varphi',s)}$	$\sigma_{(\Phi',s)}$
s_1	0.2	1
s_4	0.6667	1
s_5	0.667	1
s_6	0.333	1
s_3	1	1

(b) $\Phi' = P_{\geq 0.2}(\varphi')$



(c) MCS $\mathcal{M}_2'$

s_i	$p_{(\varphi,s)}$	$\sigma_{(\Phi,s)}$
s_1	0.1	1
s_4	0.3333	1
s_5	0.3333	1
s_6	0.6667	1
s_7	1	1

(d) $\Phi = P_{<0.1}(\varphi)$

Figure 3.3: Backward reachability for $\Phi = P_{<0.1}[P_{\geq 0.2}[tt\mathcal{U}s]\mathcal{U}t]$

3.2.1 State Formulae

True literal Let tt be the true literal. The characteristic variable $\sigma_{(tt,s)}$ is assigned 1, regardless of the label at the state s, as long as it is part of the critical subsystem.

$$\boxed{tt}$$

$$\sigma_{(tt,s)} = x_s \tag{3.22}$$

Variable	Domain	Description
$\sigma_{(\Phi,s)}$	$[0,1] \subseteq \mathbb{Z}$	The *satisfiability variable* denotes if the PCTL formula Φ is satisfiable at a state s.
x_s	$[0,1] \subseteq \mathbb{Z}$	The *inclusion variable* specifies if a state s is part of the minimal critical subsystem.
$p_{(\varphi,s)}$	$[0,1] \subseteq \mathbb{R}$	The *probability variable* assigns the probability measure $Pr(s \models \varphi)$ where φ is the path formula.
$r_s^{\rightarrow}$	$[0,1] \subseteq \mathbb{R}$	The *order variable* defines a partial order of the states for *backward reachability.*
$r_s^{\leftarrow}$	$[0,1] \subseteq \mathbb{R}$	The *order variable* defines a partial order of the states for *forward reachability.*
$t_{s,s'}^{\rightarrow}$	$[0,1] \subseteq \mathbb{Z}$	The *edge variable* denotes a transition from state s to its successor s' for *backward reachability.*
$t_{s,s'}^{\leftarrow}$	$[0,1] \subseteq \mathbb{Z}$	The *edge variable* denotes a transition from state s to its predecessor s' for *forward reachability*

Table 3.1: Variables

Atomic Formula Let $\Phi \equiv a$ be an atomic formula where $a \in AP$. The characteristic variable $\sigma_{(\Phi,s)}$ is defined with the same set of constraints as in the SMT formulation.

$$\boxed{\Phi \equiv a}$$

$$\sigma_{(\Phi,s)} = \begin{cases} 0 & \text{if } L(s) \neq a \\ x_s & \text{if } L(s) = a \end{cases} \tag{3.23}$$

Negation Let the formula $\Phi \equiv \neg\Phi'$ be the negation of a PCTL formula. This means $\sigma_{(\Phi,s)}$ needs to take complement values with respect to $\sigma_{(\Phi',s)}$ given the state s is included within the critical subsystem. It is not straightforward to translate these conditions into MILP constraints. We use a range of constraints to fix upper and lower bounds to the variables in different cases.

$$\boxed{\Phi \equiv \neg\Phi'}$$

$$\sigma_{(\Phi,s)} \leq x_s \tag{3.24a}$$

$$\sigma_{(\Phi,s)} \leq 1 - \sigma_{(\Phi',s)} \tag{3.24b}$$

$$x_s - \sigma_{(\Phi',s)} \leq \sigma_{(\Phi,s)} \tag{3.24c}$$

(3.24a) ensures that if the variable x_s is assigned 0, i.e., if state s is not a part of the critical subsystem, then $\sigma_{(\Phi,s)}$ is assigned 0. We need the remaining equations to assign $\sigma_{(\Phi,s)}$ appropriately when state s is a part of the critical subsystem. (3.24b) ensures if $\sigma_{(\Phi',s)}$ is assigned 1, then $\sigma_{(\Phi,s)}$ needs to be assigned 0. (3.24c) ensures that if $\sigma_{(\Phi',s)}$ is assigned 1 which is only possible if x_s is assigned 1, then $\sigma_{(\Phi,s)}$ is assigned 0.

Conjunction If the formula is of the form $\Phi \equiv \Phi_1 \wedge \Phi_2$, then both Φ_1 and Φ_2 need to hold. Following constraints are imposed.

$$\boxed{\Phi \equiv \Phi_1 \wedge \Phi_2}$$

$$\sigma_{(\Phi_1,s)} + \sigma_{(\Phi_2,s)} \geq 2\sigma_{(\Phi_1\wedge\Phi_2,s)} \tag{3.25a}$$

$$\sigma_{(\Phi_1,s)} + \sigma_{(\Phi_2,s)} \leq \sigma_{(\Phi_1\wedge\Phi_2,s)} + 1 \tag{3.25b}$$

(3.25a) ensures that if one of the sub-formulae is not satisfied, i.e., either $\sigma_{(\Phi_1,s)}$ or $\sigma_{(\Phi_2,s)}$ is assigned 0, then $\sigma_{(\Phi_1\wedge\Phi_2,s)}$ is assigned 0. If both the sub-formulae are assigned 1, then (3.25b) ensures that $\sigma_{(\Phi_1\wedge\Phi_2,s)}$ is assigned 1.

3.2.2 Path Formulae

A path formula φ, as discussed earlier, has to be valid for a path. If a path satisfies a path formula, then the path satisfies the constraints defining the satisfaction relation for the path formula. Let us define the MILP constraints for path formulae.

Next Path Formula Next path formula is denoted as $\varphi \equiv \bigcirc\Phi$ and the satisfaction relation is defined in 2.3.4. Following are the MILP constraints defining this satisfaction relation.

$$\boxed{\varphi \equiv \bigcirc\Phi}$$

$$\sum_{s' \in succ(s)} \sigma_{(\Phi,s')} \geq \sigma_{(\bigcirc\Phi,s)} \tag{3.26a}$$

$$x_s \geq \sigma_{(\bigcirc\Phi,s)} \tag{3.26b}$$

$$\frac{1}{|succ(s)|} \cdot \sum_{s' \in succ(s)} \sigma_{(\Phi,s')} + x_s \leq \sigma_{(\bigcirc\Phi,s)} + 1 \tag{3.26c}$$

(3.26a) ensures if the summation of $\sigma_{(\Phi,s')}$ over all the successors of s is 0, i.e. there is not a single successor s' of s that satisfies Φ, then the next path formula is not satisfied and $\sigma_{(\bigcirc\Phi,s)}$ is assigned 0. (3.26b) ensures that if a state s is not a part of the critical subsystem, then the path formula is not satisfied at s. (3.26c) ensures that if there exists at least one successor s' of s that satisfies Φ, then $\sigma_{(\bigcirc\Phi,s)}$ is assigned 1 given s is part of the critical subsystem. In case a state s has multiple successors where Φ holds, then $\frac{1}{|succ(s)|} \cdot \sum_{s' \in succ(s)} \sigma_{(\Phi,s')} > 0$. If s is part of the critical subsystem, then x_s is assigned 1. This means the LHS of (3.26c) greater than 1. This forces the RHS to be greater than 1 since it is the upper bound assigning 1 to $\sigma_{(\bigcirc\Phi,s)}$.

P Formula for Next Path Formula Let the P formula be denoted as $P_{\bowtie\lambda}(\bigcirc\Phi)$. The following constraints apply.

$$\boxed{\Psi \equiv P_{\bowtie\lambda}(\bigcirc\Phi)}$$

$$p_{(\bigcirc\Phi,s)} \leq \sigma_{(\bigcirc\Phi,s)} \tag{3.27a}$$

$$p_{(\bigcirc\Phi,s)} \leq x_s \tag{3.27b}$$

$$p_{(\bigcirc\Phi,s)} \leq \sum_{s' \in succ(s)} P(s,s') \cdot \sigma_{(\Phi,s')} \tag{3.27c}$$

(3.27a) ensures that $p_{(\bigcirc\Phi,s)}$ is assigned 0 if $\bigcirc\Phi$ is not satisfied at the state s. (3.27b) ensures that $p_{(\bigcirc\Phi,s)}$ is assigned 0 if s is not part of the critical subsystem. (3.27c) assigns probability value to the state s by summing up the transition probabilities from the state s to all its successor states that satisfy Φ.

Case: $\bowtie$ is $\leq$ λ here is a constant. It is the bound specified in the property. If $\lambda = 1$, then the probability value cannot violate the bound. Hence, $\sigma_{(P_{\leq\lambda}(\bigcirc\Phi),s)}$ is always assigned 1 in (3.28a). Otherwise, (3.28b) ensures that $\sigma_{(P_{\leq\lambda}(\bigcirc\Phi),s)}$ is assigned 1 if $\lambda - p_{(\bigcirc\Phi,s)} \in [0,1) \subseteq \mathbb{R}$. Hence, (3.28a) and (3.28b) combined assign to $\sigma_{(P_{\leq\lambda}(\bigcirc\Phi),s)}$ value 1 if the bound is fulfilled. (3.28c) assigns to $\sigma_{(P_{\leq\lambda}(\bigcirc\Phi),s)}$ value 0 if the bound is violated.

$$\boxed{\Psi \equiv P_{\leq\lambda}(\bigcirc\Phi)}$$

$$\sigma_{(\Phi,s)} = 1 \quad \text{iff } \lambda = 1 \tag{3.28a}$$

$$\lambda - p_{(\bigcirc\phi,s)} < \sigma_{(\Phi,s)} \quad \text{iff } \lambda \in [0,1) \subseteq \mathbb{R} \tag{3.28b}$$

$$\sigma_{(\Phi,s)} \leq \lambda - p_{(\bigcirc\Phi,s)} + 1 \tag{3.28c}$$

Case: $\bowtie$ is $\geq$ If $\lambda = 0$ and the condition that needs to hold is $p \geq \lambda$, then no probability can violate the bound. Hence (3.29a) assigns $\sigma_{(P_{\leq\lambda}(\bigcirc\Phi),s)}$ value 1. Otherwise, (3.29b) assigns $\sigma_{(P_{\leq\lambda}(\bigcirc\Phi),s)}$ value 1 if the bound is satisfied. (3.29c) ensures $\sigma_{(P_{\leq\lambda}(\bigcirc\Phi),s)}$ is assigned value 0 if the bound is not satisfied.

$$\boxed{\Psi \equiv P_{\geq\lambda}(\bigcirc\Phi)}$$

$$\sigma_{(\Phi,s)} = 1 \quad \text{iff } \lambda = 0 \tag{3.29a}$$

$$-\lambda + p_{(\bigcirc\phi,s)} < \sigma_{(\Phi,s)} \quad \text{iff } \lambda \in (0,1] \subseteq \mathbb{R} \tag{3.29b}$$

$$\sigma_{(\Phi,s)} \leq -\lambda + p_{(\bigcirc\Phi,s)} + 1 \tag{3.29c}$$

Remark. *It has to be noted that the inner P formulae always have upper bounds while the outer P formulae only have lower on probability values.*

Remark. *Since MILP solvers make use of floating-point arithmetic, there may be rounding errors. Though there are hardly any state-of-art model checkers that use exact arithmetic [WKHB07]. The impact of rounding errors is assumed to be negligible.*

Until Path Formula Until path formula can be denoted as $\varphi \equiv \Phi_a \mathcal{U} \Phi_b$. Here are the MILP constraints.

$$\boxed{\varphi \equiv \Phi_a \mathcal{U} \Phi_b}$$

$$\sigma_{(\Phi_a,s)} + \sigma_{(\Phi_b,s)} \geq \sigma_{(\varphi,s)} \tag{3.30a}$$

$$\sigma_{(\Phi_b,s)} \geq \sigma_{(\varphi,s)} - \sum_{s' \in succ(s) \setminus \{s\}} \sigma_{(\varphi,s')} \tag{3.30b}$$

$$\sigma_{(\Phi_b,s)} \leq \sigma_{(\varphi,s)} \tag{3.30c}$$

(3.30a) says $\sigma_{(\varphi,s)}$ will be assigned 0 if neither Φ_a nor Φ_b hold at a state s. (3.30b) says that if $\sigma_{(\varphi,s)}$ is assigned 1 at the state s, then there exists at least one successor s' of s, excluding self-loops, that satisfies $\sigma_{(\varphi,s')}$. The only exception to this rule is if the state s satisfies Φ_b. (3.30b) leads to a chain of implications that ensure reachability to Φ_b. Though (3.30b) handles self-loops, it cannot handle larger loops and requires backward reachability constraints. (3.30c) is a redundant constraint that cuts the search space.

Backward reachability for inner until formula Backward reachability is needed to ensure that every path that is considered to satisfy an until path formula $\varphi \equiv \Phi_a \mathcal{U} \Phi_b$ will end at a Φ_b-state.

$$\boxed{\varphi \equiv \Phi_a \mathcal{U} \Phi_b}$$

$$\forall s \in S, \forall s' \in succ(s): \; 2t^{\rightarrow}_{s,s'} \leq \sigma_{(\varphi,s)} + \sigma_{(\varphi,s')} \tag{3.31a}$$

$$r^{\rightarrow}_{s} < r^{\rightarrow}_{s'} + (1 - t^{\rightarrow}_{s,s'}) \tag{3.31b}$$

$$-\sigma_{(\varphi,s)} + \sum_{s' \in succ(s) \setminus \{s\}} t^{\rightarrow}_{s,s'} \geq -\sigma_{(\Phi_b,s)} \tag{3.31c}$$

Let us consider a relevant path here to be a path that satisfies $\Phi_a \mathcal{U} \Phi_b$. (3.31a) enforces that if either a state s or its successor s' are not a relevant path, then the variable $t^{\rightarrow}_{s,s'}$ denoting a transition from s to s' is set to 0. (3.31b) ensures that if there is a transition between two states s and s', then the condition $r^{\rightarrow}_{s} < r^{\rightarrow}_{s'}$ needs to hold. This ensures the assignment does not terminate at a loop. (3.31c) ensures that every state s on the relevant path needs to have an outgoing transition to a state s' unless the state s is a Φ_b-state in which case it is the last state of the relevant path.

Bounded Until Path Formula Bounded until path formula is denoted as $\Phi_a \mathcal{U}^{\leq n} \Phi_b$. Here are the additional constraints for bounded until that need to be satisfied within n transitions.

$$\boxed{\Phi \equiv \Phi_a \mathcal{U}^{\leq n} \Phi_b}$$

$$\sum_{s \in S, s' \in pred(s)} t^{\leftarrow}_{s,s'} \leq n \tag{3.32a}$$

$$\sum_{s \in S, s' \in succ(s)} t^{\rightarrow}_{s,s'} \leq n \tag{3.32b}$$

$t^{\leftarrow}_{s,s'}$ are edge variables to keep track of forward reachability and $t_{s,s'}$ keeps track of backward reachability. (3.32a) is applied for an inner bounded until formula since there is no formulation of forward reachability for inner formula. This is because a path that satisfies an inner until formula need not have the initial state as a part of it. It has other paths that start from the initial state to merge at some point.

P Formula for Until Path Formula Until path formula is denoted as $\Phi \equiv P_{\bowtie\lambda}(\Phi_a \mathcal{U} \Phi_b)$. Following are the MILP constraints.

$$\boxed{\Phi \equiv P_{\bowtie\lambda}(\Phi_a \mathcal{U} \Phi_b)}$$

$$p_{(\varphi,s)} \leq \sigma_{(\varphi,s)} \tag{3.33a}$$

$$\sigma_{(\Phi_b,s)} \leq p_{(\varphi,s)} \tag{3.33b}$$

$$p_{(\varphi,s)} \leq x_s \tag{3.33c}$$

$$p_{(\varphi,s)} \leq \sum_{s' \in succ(s)} P(s,s') \cdot p_{(\varphi,s')} + 1 + \sigma_{(\Phi_b,s)} - \sigma_{(\Phi_a,s)} \tag{3.33d}$$

Here $\varphi \equiv \Phi_a \mathcal{U} \Phi_b$. (3.33a) ensures that if a state s satisfies neither Φ_a nor Φ_b, then $p_{(\varphi,s)}$ is assigned 0. This means it is not on a relevant path. Let us now consider (3.33b). If a state s satisfies Φ_b, then $\sigma_{(\Phi_b,s)}$ is assigned 1. (3.33b) ensures that $p_{(\varphi,s)}$ is assigned 1. If a state s does not satisfy Φ_b, then $\sigma_{\Phi_b,s}$ is assigned 0 and $p_{(\varphi,s)} \in [0,1] \subseteq \mathbb{R}$. (3.33c) ensures that if a state s is not included in the critical system (i.e., x_s is assigned 0), then the probability of the state satisfying the formula φ is 0. (3.33d) assigns an upper-bound for the probability of Φ_a-states. If a state satisfies both Φ_a and Φ_b, the upper bound is disabled in (3.33d).

Case: $\bowtie$ is $\leq$

$$\boxed{\Phi \equiv P_{\leq\lambda}(\Phi_a \mathcal{U} \Phi_b)}$$

$$\sigma_{(\Phi,s)} = 1 \qquad \text{iff } \lambda = 1 \tag{3.34a}$$

$$\lambda - p_{(\varphi,s)} < \sigma_{(\Phi,s)} \qquad \text{iff } \lambda \in [0,1) \subseteq \mathbb{R} \tag{3.34b}$$

$$\sigma_{(\Phi,s)} \leq \lambda - p_{(\varphi,s)} + 1 \tag{3.34c}$$

Case: $\bowtie$ is $\geq$

$$\boxed{\Phi \equiv P_{\geq\lambda}(\Phi_a \mathcal{U} \Phi_b)}$$

$$\sigma_{(\Phi,s)} = 1 \qquad \text{iff } \lambda = 0 \tag{3.35a}$$

$$-\lambda + p_{(\varphi,s)} < \sigma_{(\Phi,s)} \qquad \text{iff } \lambda \in (0,1] \subseteq \mathbb{R} \tag{3.35b}$$

$$\sigma_{(\Phi,s)} \leq -\lambda + p_{(\varphi,s)} + 1 \tag{3.35c}$$

3.2.3 Minimal Critical Subsystem

Given a property of form $P_{\leq}[\varphi]$, we need to compute a minimal critical subsystem that violates it. Which means the initial state s_I needs to violate the bound. This is more straightforward with MILP formulation. We need an objective function to get minimum possible states that violate the bound. Hence, inclusion variables x_s can be a part of the objective function. Computation of MCS can be summarized with following constraints.

$$\text{minimize } \alpha p_{(\varphi,s_I)} + \sum_{s \in S} x_s \tag{3.36a}$$

such that

$$p_{(\varphi,s_I)} > \lambda \tag{3.36b}$$

MILP solvers can handle objective function and hence the above constraints can be handled by the solver. We need to maximize $p_{(\varphi,s_I)}$ because probability constraints only have an upper bound and do no assign actual probabilities to the states. Since every state within the MCS is reachable from the initial state, maximizing initial probability $p_{(\varphi,s_I)}$ would enforce the solver to assign actual probabilities to the states. Following is the domain of the co-efficient of $p_{(\varphi,s_I)}$, i.e. $\alpha \in (-1,0) \subseteq \mathbb{R}$. We ensure the co-efficient is negative but greater than -1, so that $p_{(\varphi,s_I)}$ is maximized but is not confused for an additional state.

3.2.4 Optimizations

We provide a few optimizations that are redundant constraints which decrease the search space for the solver resulting in speeding up of the search for the optimal solution.

Forward Reachability Forward reachability constraints ensure that every state that has been assigned needs to be reachable from the initial state. Minimizing the number of states will ensure that any state that does not contribute to the satisfiability of any formula within the critical subsystem will be unallocated. However, these set of constraints further assist the solver in making the assignments and can be seen as an optimization to reduce the search space.

$$\boxed{\varphi \equiv \Phi_a \mathcal{U} \Phi_b}$$

$$\forall s \in S, \forall s' \in pred(s) : \; 2t^{\leftarrow}_{s,s'} \leq x_s + x_{s'} \tag{3.37a}$$

$$r^{\leftarrow}_{s} < r^{\leftarrow}_{s'} + (1 - t^{\leftarrow}_{s,s'}) \tag{3.37b}$$

$$\forall s \in S \setminus \{s_I\} : \; -x_s + \sum_{s' \in pred(s) \setminus \{s\}} t^{\leftarrow}_{s,s'} \geq 0 \tag{3.37c}$$

Here, $pred(s)$ is a *predecessor function* that returns the set of predecessors of s, $t_{s,s'}$ is the edge variable denoting a transition between s and s' and $r^{\leftarrow}_{s}$ is an order variable that ensures that the assignment does not end at a loop. s_I is the initial state.

Forward Cuts Forward cut is a predecessor constraint that ensures that if a state s is a part of the critical subsystem, then it has at least one predecessor $s' \in pred(s)$, where $pred(s)$ returns the set of predecessors of s, is part of the critical subsystem. This constraint applies to all states excluding the initial state s_I. It ensures that a state or path fragment unreachable from the initial state is not assigned. These set of constraints are less expensive than the forward reachability constraints since unlike forward reachability, it does not have a chain of implications.

$$\forall s \in S \setminus \{s_I\} : \; -x_s + \sum_{s' \in pred(s) \setminus \{s\}} x'_s \geq 0 \tag{3.38}$$

Satisfiability cuts A property in PCTL can be composed of many sub-formulae. A state s cannot be a part of the critical subsystem if it is not part of a satisfiability set of at least one sub-formula.

$$x_s \leq \sum_{0 \leq i \leq n} \sigma_{(\psi_i, s)} \tag{3.39}$$

ψ here can be either a state formula or a path formula. n is the number of sub-formulae.

Chapter 4

Example and Experiments

In this chapter, we demonstrate how the FullPctlSubSys tool generates a linear constraint system with the MILP formulation. We later analyse certain case studies available at the homepage of PRISM [KNP+12].

4.1 Example

Let us consider the Example 2.5.1 in Section 2.5. Figure 2.1 is the DTMC we are model checking. The PCTL property in consideration is $\Phi = P_{\leq 0.2}[P_{>0.5}[tt\mathcal{U}a]\,\mathcal{U}a]$. We generate a parse tree for the property, similar to Figure 2.2(a). Let us denote the sub-formulae with the notations given in (2.2) in Section 2.3. We do a bottom up traversal of the tree. We start with the leaf node labelled tt which is the true-literal. Based on (3.22), we can generate constraints for all the states in 2.1.

$$\boxed{tt}$$

$$\sigma_{(tt,s_1)} = x_{s_1}$$

$$\vdots$$

$$\sigma_{(tt,s_6)} = x_{s_6}$$

We now move to the node labelled a which is an atomic formula with the atomic proposition $\{a\}$. Based on (3.23), we check the label at every state based on which we add the constraint. Hence, we add the following constraints to the stack.

$$\boxed{a}$$

$$\begin{aligned}\sigma_{(a,s_1)} &= 0\\ \sigma_{(a,s_2)} &= 0\\ \sigma_{(a,s_3)} &= 0\\ \sigma_{(a,s_4)} &= x_{s_4}\\ \sigma_{(a,s_5)} &= 0\\ \sigma_{(a,s_6)} &= 0\end{aligned}$$

We now traverse to the parent node of tt and a which is labelled $\mathcal{U}$. This is the until path formula that is built based on the satisfiability sets of its child nodes. In accordance with equations (3.30a)-(3.30c), we add the following constraints. In (3.30b), we do a summation of σ variables of the successors of a state excluding any self-loops. Hence, s_6 has no successors.

$$\boxed{\varphi' = tt\mathcal{U}a}$$

$$\begin{aligned}\sigma_{(tt,s_1)} + \sigma_{(a,s_1)} &\geq \sigma_{(\varphi',s_1)}\\ \sigma_{(a,s_1)} &\geq \sigma_{(\varphi',s_1)} - \sigma_{(\varphi',s_2)}\\ \sigma_{(a,s_1)} &\leq \sigma_{(\varphi',s_1)}\\ &\vdots\\ \sigma_{(tt,s_6)} + \sigma_{(a,s_6)} &\geq \sigma_{(\varphi',s_6)}\\ \sigma_{(a,s_6)} &\geq \sigma_{(\varphi',s_6)}\\ \sigma_{(a,s_6)} &\leq \sigma_{(\varphi',s_6)}\end{aligned}$$

We can assign backward reachability constraints for the until path formula. This ensures backward reachability to an a-state for the until formula $tt\mathcal{U}a$. We add the following constraints based on equations (3.31a)-(3.31c). We need to note that (3.31b) has a strict inequality. The MILP solver we are using cannot solve a strict inequality. A strict inequality $a < b$ is converted to a non-strict inequality $a + \epsilon \leq b$ where ϵ is a small configurable tolerance of the order 10^{-6}.

$$\boxed{\varphi' = tt\mathcal{U}a}$$

$$\begin{aligned}2t^{\rightarrow}_{s_1,s_2} &\leq \sigma_{(\varphi',s_1)}\sigma_{(\varphi',s_2)}\\ r^{\rightarrow}_{s_1} + \epsilon &\leq r^{\rightarrow}_{s_2} + (1 - t^{\rightarrow}_{s_1,s_2})\\ -\sigma_{(\varphi',s_1)} + t^{\rightarrow}_{s_1,s_2} &\geq -\sigma_{(a,s_1)}\\ &\vdots\end{aligned}$$

Continuing this process, we go up the tree and add constraints for the nodes labelled $P_{>0.5}$. We encounter another node labelled a, a part of the path formula $\varphi'' = \psi' \mathcal{U} a$. We can reuse the variables from the first node labelled a, which is part of the path formula $\varphi' = tt\mathcal{U}a$. After doing a complete traversal of the parse tree, we add a set of constraints for MCS. To violate the property Φ, we need to build a subsystem with a set of states that satisfy the outer path formula φ'' and contribute to probability that is enough to exceed the bound ($\lambda = 0.25$). Based on (3.36a) and (3.36b), we add the following constraints.

$$\boxed{\text{MCS}}$$

$$p_{(\varphi'',s_1)} \geq 0.25 + \epsilon$$

$$\text{minimize} \quad -0.5 \cdot p_{(\varphi'',s_1)} + x_{s_1} + x_{s_2} + x_{s_3} + x_{s_4} + x_{s_5} + x_{s_6}$$

The strict inequality in (3.36b) has been changed to a non-strict inequality and α in (3.36a) is set to -0.5. The assumption while adding the constraints for MCS is that the property in consideration is violated by the DTMC. In case the DTMC satisfies the property, the MILP solver returns that the model is infeasible. If we run this model with one of the MILP solvers, *Gurobi* [Opt12] for instance, we get a solution summarized in the Table 4.1.

We can see that the satisfiability variable for the property in question, i.e., $\sigma_{(\Phi,s_i)}$, on the penultimate column is assigned 0 for every state and more importantly for the initial state s_1. Hence, the property is violated by the DTMC. The counterexample is an MCS with 5 states i.e., all the states which have x_s assigned 1. The probability of satisfying the outermost until formula φ'' at state s_1 is 0.5 which violates the upper bound 0.25. The constraints for assigning probability values only impose an upper bound. Hence, we include the probability variable $p_{(\varphi'',s_1)}$ within the objective function in order to get maximum probabilities. An interesting column to observe is the $5th$ column which contains probability values assigned for the variable $p_{(\varphi',s_i)}$. Here, $\varphi' = tt\mathcal{U}a$ is an inner until formula. It is wrapped by the P formula $P_{>0.5}[\varphi']$ which means the probability values at every state needs to exceed 0.5 to satisfy the P formula. Table 4.2 shows the probability values assigned and the actual probabilities for the states within the MCS. We can see that the solver ensures that in the case the states satisfies the bound, it assigns probability value higher than 0.5. However, for the state s_5 which does not satisfy the bound, it assigns an arbitrary value of lower than 0.5. Though the maximal probability values are not assigned, the solver ensures that it assigns the value that lies on the correct side of the bound.

i	$\sigma_{(tt,s_i)}$	$\sigma_{(a,s_i)}$	$\sigma_{(\varphi',s_i)}$	$p_{(\varphi',s_i)}$	$\sigma_{(\Psi',s_i)}$	$\sigma_{(\varphi'',s_i)}$	$p_{(\varphi'',s_i)}$	$\sigma_{(\Phi,s_i)}$	x_{s_i}
1	1	0	1	0.51	1	1	0.5	0	1
2	1	0	1	0.51	1	1	0.5	0	1
3	1	0	1	0.51	1	1	0.5	0	1
4	1	1	1	1	1	1	1	0	1
5	1	0	1	0.02	0	0	0	0	1
6	0	0	0	0	0	0	0	0	0

Table 4.1: Optimal Solution for the model

i	$p_{(\varphi',s_i)}$	$Pr(s_i \models \varphi')$	x_{s_i}
1	0.51	0.75	1
2	0.51	0.75	1
3	0.51	0.75	1
4	1	1	1
5	0.02	0.5	1
6	0	0	0

Table 4.2: Assigned and maximal probabilities of satisfying φ'

4.2 Experiments

We do the experimental evaluation of the SMT and MILP formulation for PCTL properties. We generate MCS for randomized protocols in case the property is refuted. The computer used to run the experiments has a dual core Intel Core i5 M430 processor with a clock speed of 2.27 GHz, a memory of 2.6 GB running Ubuntu 12.04 64-bit operating system. A tool named FullPctlSubSys has been developed in C++. It can parse a nested PCTL property, create a parse tree, add constraints and instantiate a solver to solve the linear constraint system. The *Microsoft Z3 solver* is used as the SMT solver [DMB08]. The *Gurobi solver* is used as the MILP solver [Opt12]. We use the respective solver APIs to call the solver. We export the DTMC models using PRISM. The files generated by PRISM are provided as an input to the tool [KNP+12]. The following randomized protocols are used for benchmarking the formulations.

The Crowds Protocol The crowds protocol provides anonymity to a user surfing internet providing a greater privacy for web transactions [RR98]. The idea is to route the request to a server through a random set of nodes within the pool of users or the *crowd*. Each time a sender sends a message, it selects another member in the crowd at random which includes the sender itself. The message is encrypted with a pairwise key. The selected router flips a biased coin. It sends the message directly to the server with a probability $1-p_f$ where where p_f is the forwarding probability. It sends the message to another crowd member with p_f probability where p_f is the forwarding probability. The route is fixed after the first message. There are corrupt members that are trying to track the sender of the message. The corrupt member has some visibility to the routing of the message if only if it is a part of the route. The corrupt member can only know who is the previous sender from whom it receives the message but should not be able tell if the previous sender is the original sender of the message. The protocol gives varying degrees of anonymity based on system parameters such as the size of the crowd, the percentage of corrupt members. One of the degrees of anonymity is called *probable innocence.* With probable innocence, a corrupt member cannot say with a probability more than 0.5 if a crowd member is an original sender.

New members join and old members fail. This calls for new paths. New paths lead to considerable degradation of anonymity as opposed to static paths [Shm02]. This leads to increasing probability of detecting the path initiator called the *Positive* event. This is tackled by increasing the crowd size. Though this results in decreasing probability of reaching the *Positive* event, it leads to an increased confidence for the observations of the adversary. We are interested in states where the adversary has observed the real sender and only the real sender. This event is called *Confidence.* We want to reach this state with a probability of no more than λ. This property can be specified in PCTL accordingly, $P_{\leq\lambda}[tt\,\mathcal{U}\,\mathit{Positive} \wedge \mathit{Confidence}]$.

The model simulating the crowds protocol can vary based on crowd size (N) and the number of path reformulations (R). It is denoted by *crowds-N-R.* Let us consider the Table 4.3. The first column lists the *model* in consideration, followed by the number of states ($|S|$) in the model, the number of transitions ($|E|$), the upper probability bound (λ) to be violated, the number of states in the MCS ($|S_{MCS}|$) as calculated by the solvers.

Let us consider the Table 4.4 which lists the running times for the models. The first column lists the model in consideration, followed by the running times by the Z3 solver. The first column within the SMT section is running the benchmark without any optimizations, followed by enabling bound cuts, followed by enabling all the remaining optimizations, i.e., forward reachability, forward cuts and satisfiability cuts. We can observe that the bound cuts drastically reduce the running times. The reduction is 30% for crowds2-3 and 57% for crowds2-4 and crowds2-5. Applying all the optimizations shows a reduction of 64% running time for crowds2-3 and 57% for crowds3-3 saving upto 6390 seconds. However, applying all the optimizations may not always benefit as seen for the crowds2-4.

We can see that the MILP formulation is much more efficient. The MILP solvers have a running time that is a fraction of that of the SMT solvers. This can be attributed to the absence of disjunctive constraints within the MILP formulation. The first column within the MILP section is running the benchmark with no optimizations, the second column applies all optimizations and the third column considers the best configuration of the optimizations. It was observed that applying all optimizations turned out to be more expensive and often resulted in greater running times. Therefore, a subset of optimizations that yields the best result has been considered. Though the redundant constraints do not reduce the running time for models with N=2, a combination of forward cuts and satisfiability cuts causes a 40% reduction in the running time in crowds3-5 and

Model	$\lvert S\rvert$	$\lvert E\rvert$	λ	$\lvert S_{MCS}\rvert$
crowds2-3	183	243	0.05	29
crowds2-4	356	476	0.05	29
crowds2-5	612	822	0.05	29
crowds3-3	396	576	0.05	52
crowds3-4	901	1321	0.05	52
crowds3-5	1772	2612	0.05	52
crowds5-4	3515	6035	0.05	99
crowds5-6	18817	32677	0.05	99

Table 4.3: Sizes of the crowds benchmark models

Model	SMT Solver Z3			MILP Solver Gurobi		
	No Opt.	Bound cut	All Opt.	No Opt.	All Opt.	Best conf.
crowds2-3	31.16	21.53	7.64	0.19	0.29	0.19
crowds2-4	178.25	75.40	117.34	0.50	0.80	0.50
crowds2-5	491.37	207.94	206.01	1.03	1.50	1.03
crowds3-3	TL	11022.5	4632.46	0.71	1.00	0.59
crowds3-4	TL	TL	TL	15	6.76	5.45
crowds3-5	TL	TL	TL	30.47	83.95	18.68
crowds5-4	TL	TL	TL	36.66	161.08	36.66
crowds5-6	TL	TL	TL	710.70	1328.65	321.07

Table 4.4: Running times for the crowds benchmark models in seconds

a 50% reduction in crowds5-6.

Bounded Retransmission protocol Bounded transmission protocol (BRP) has a *sender* and a *receiver* communicating with each other through two lossy unreliable channels [DJJL01]. The sender transmits the file in N chunks and each chunk can be transmitted for a maximum of K times in case of a failure in transmission. The property in consideration is that the receiver does not receive any message ($\neg recv@receiver$) and the sender has tried to send a message and is not in an idle state ($\neg idle@sender$). This corresponds to property 2 in [DJJL01]. This can be specified in PCTL in the following way, $P_{<=\lambda}[\,tt\,\mathcal{U}\,\neg idle@sender \wedge \neg recv@receiver]$. We will check many models denoted by *brp-N-K* with different N and K values. From running the experiments for crowds protocol, we realized that applying all optimizations for SMT does not always benefit. Hence, we perform experiments with the Z3 SMT solver to evaluate the various optimizations within the SMT formulation.

Table 4.5 lists the name of the model, the number of states,the number of transitions, the upper probability bound and the size of minimal critical subsystems calculated. Given the small size of the model and the very low probability bounds, this model is appropriate to test the behaviour of the SMT formulation due to the high running times of SMT. Table 4.6 lists the running times of

Model	$\|S\|$	$\|E\|$	λ	$\|S_{MCS}\|$
brp16-2	677	867	$7.000000000000001E-6$	9
brp16-3	886	1155	$0.6000000000000003E-7$	11
brp16-4	1095	1443	$2.2000000000000005E-9$	13
brp16-5	1034	1731	$5.400000000000001E-11$	15

Table 4.5: Sizes of the benchmark models for BRP

Model	SMT Solver Z3					
	Bound cut	Sat. cut	Fwd cut	Fwd reach	All Opt.	Best conf.
brp16-2	13.78	11.58	11.42	12.17	11.30	11.241
brp16-3	28.93	23.99	23.51	26.13	23.51	22.22
brp16-4	58.63	49.93	57.26	53.60	51.30	49.93
brp16-5	93.48	86.37	86.36	89.61	85.19	85.19

Table 4.6: Running times for the benchmark models in seconds

the formulation. The first column has only bound cut enabled, the rest of the columns have different optimizations enabled coupled with bound cuts since we are interested in the effect of other optimizations. The second column lists the effect of satisfiability cuts, followed by forward cuts, followed by forward reachability cuts, followed by all optimizations and then comes the best configuration. In case of brp16-2 and brp16-3, the best configuration was a combination of satisfiability and forward cuts. In brp16-4, satisfiability cut outperformed all other optimizations. In brp16-5, all optimizations together perform better, however satisfiability and forward cuts come at a close second. The optimizations that are most likely to benefit the most are satisfiability cuts and forward cuts. Forward reachability turns out to be an expensive optimization probably because of the chain of implications.

Chapter 5

Conclusion

In this work, we have presented methods to compute optimal counterexamples in form of state-minimal critical subsystems for full PCTL properties. This involves reachability, bounded reachability and constrained reachability and other nested PCTL formulae. We provide both SMT and MILP formulation for DTMCs. The method involves building linear constraints for the satisfiability of every sub-formula in the property along with building constraints for the violation of the bound of the outer formula. The MILP formulation clearly outperforms the SMT formulation. The MILP formulation is manifold faster. We provide many optimizations such as forward reachability, forward cuts, satisfiability cuts and bound cuts that considerably speed up the running time. The experimental evaluation reveals that the optimization applied on the SMT formulation results in a best case reduction of 64% in the running time. The optimizations on MILP show a best case reduction of 50% in the running time. Bound cuts drastically reduce running times for SMT while a combination of forward cuts and satisfiability cuts do the trick for MILP.

Future work The formulation can be extended for Markov decision processes. There needs to be a theoretical investigation on the complexity of computing MCS for DTMCs [WJÁ$^+$12b]. Given certain optimizations such as backward reachability and backward cuts cannot be applied while calculating MCS for full PCTL properties, we need to formulate other optimizations.

Bibliography

[AL10] Husain Aljazzar and Stefan Leue. Directed explicit state-space search in the generation of counterexamples for stochastic model checking. *Software Engineering, IEEE Transactions on*, 36(1):37–60, 2010.

[BK08] C. Baier and J.P. Katoen. *Principles of Model Checking*. 2008.

[BP12] Simon Busard and Charles Pecheur. Rich counter-examples for temporal-epistemic logic model checking. *arXiv preprint arXiv:1202.4509*, 2012.

[CGJ$^+$03] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM (JACM)*, 50(5):752–794, 2003.

[CGMZ95] Edmund M Clarke, Orna Grumberg, Kenneth L McMillan, and Xudong Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proceedings of the 32nd annual ACM/IEEE Design Automation Conference*, pages 427–432. ACM, 1995.

[CJLV02] Edmund Clarke, Somesh Jha, Yuan Lu, and Helmut Veith. Tree-like counterexamples in model checking. In *Logic in Computer Science, 2002. Proceedings. 17th Annual IEEE Symposium on*, pages 19–29. IEEE, 2002.

[Cla08] Edmund M Clarke. The birth of model checking. In *25 Years of Model Checking*, pages 1–26. Springer, 2008.

[CV10] Rohit Chadha and Mahesh Viswanathan. A counterexample-guided abstraction-refinement framework for markov decision processes. *ACM Transactions on Computational Logic (TOCL)*, 12(1):1, 2010.

[DDM06] Bruno Dutertre and Leonardo De Moura. Integrating simplex with dpll (t). *CSL, SRI INTERNATIONAL, Tech. Rep*, 2006.

[DJJL01] Pedro R DâĂŹArgenio, Bertrand Jeannet, Henrik E Jensen, and Kim G Larsen. Reachability analysis of probabilistic systems by successive refinements. In *Process Algebra and Probabilistic Methods. Performance Modelling and Verification*, pages 39–56. Springer, 2001.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[GMZ04] Paul Gastin, Pierre Moro, and Marc Zeitoun. Minimization of counterexamples in spin. In *Model Checking Software*, pages 92–108. Springer, 2004.

[HKB09] Tingting Han, J-P Katoen, and Damman Berteun. Counterexample generation in probabilistic model checking. *Software Engineering, IEEE Transactions on*, 35(2):241–257, 2009.

[HWZ08] Holger Hermanns, Björn Wachter, and Lijun Zhang. Probabilistic cegar. In *Computer Aided Verification*, pages 162–175. Springer, 2008.

[IR90] Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1):60–87, 1990.

[JÁK+11] Nils Jansen, Erika Ábrahám, Jens Katelaan, Ralf Wimmer, Joost-Pieter Katoen, and Bernd Becker. Hierarchical counterexamples for discrete-time markov chains. In *Automated Technology for Verification and Analysis*, pages 443–452. Springer, 2011.

[KNP+12] Marta Kwiatkowska, Gethin Norman, David Parker, et al. The prism benchmark suite. In *9th International Conference on Quantitative Evaluation of SysTems*, pages 203–204, 2012.

[KSB08] D. Kröning, O. Strichman, and R. Bryant. *Decision Procedures: An Algorithmic Point of View.* Texts in Theoretical Computer Science. Springer-Verlag Berlin Heidelberg, 2008.

[Nel95] R. Nelson. *Probability, Stochastic Processes, and Queueing Theory: The Mathematics of Computer Performance Modeling.* 1995.

[Opt12] Gurobi Optimization. Gurobi optimizer reference manual. *URL: http://www. gurobi. com*, 2012.

[RR98] Michael K Reiter and Aviel D Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security (TISSEC)*, 1(1):66–92, 1998.

[SB05] Viktor Schuppan and Armin Biere. *Shortest counterexamples for symbolic model checking of LTL with past.* Springer, 2005.

[Shm02] Vitaly Shmatikov. Probabilistic analysis of anonymity. In *Computer Security Foundations Workshop, 2002. Proceedings. 15th IEEE*, pages 119–128. IEEE, 2002.

[ST04] Daniel A Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM (JACM)*, 51(3):385–463, 2004.

[Van08] R.J. Vanderbei. *Linear programming.* International Series in Operations Research & Management Science, v. 114. Springer London, Limited, 2008.

[WJÁ$^+$12a] Ralf Wimmer, Nils Jansen, Erika Ábrahám, Bernd Becker, and Joost-Pieter Katoen. Minimal critical subsystems for discrete-time markov models. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 299–314. Springer, 2012.

[WJÁ$^+$12b] Ralf Wimmer, Nils Jansen, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. Minimal counterexamples for refuting ω-regular properties of Markov decision processes. Reports of SFB/TR 14 AVACS 88, September 2012. ISSN: 1860-9821, `http://www.avacs.org`.

[WKHB07] Ralf Wimmer, Alexander Kortus, Marc Herbstritt, and Bernd Becker. Symbolic Model Checking for DTMCs with Exact and Inexact Arithmetic. Reports of SFB/TR 14 AVACS 30, August 2007. ISSN: 1860-9821, `http://www.avacs.org`.