Statistical and Logical Methods for Property Checking

by

Przemysław Daca

January 2, 2017

A thesis presented to the Graduate School of the
Institute of Science and Technology Austria, Klosterneuburg, Austria
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

The thesis of Przemysław Daca, titled *Statistical and Logical Methods for Property Checking*, is approved by:

Supervisor: Thomas A. Henzinger, IST Austria, Klosterneuburg, Austria

Signature: _____

Committee Member: Dirk Beyer, Ludwig-Maximilian University of Munich, Germany

Signature: _____

Committee Member: Dejan Ničković, AIT Austrian Institute of Technology, Vienna, Austria

Signature: _____

Committee Member: Georg Weissenbacher, TU Wien, Vienna, Austria

Signature: _____

Defense Chair: Björn Hof, IST Austria, Klosterneuburg, Austria

Signature: _____

I hereby declare that this thesis is my own work and that it does not contain other people's work without this being so stated; this thesis does not contain my previous work without this being stated, and the bibliography contains all the literature that I used in writing the dissertation.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other university or institution.

I certify that any republication of materials presented in this thesis has been approved by the relevant publishers and co-authors.

Signature: _____

Przemysław Daca

January 2, 2017

To Ilona and my parents.

Abstract

This dissertation concerns the automatic verification of probabilistic systems and programs with arrays by statistical and logical methods. Although statistical and logical methods are different in nature, we show that they can be successfully combined for system analysis.

In the first part of the dissertation we present a new statistical algorithm for the verification of probabilistic systems with respect to unbounded properties, including linear temporal logic. Our algorithm often performs faster than the previous approaches, and at the same time requires less information about the system. In addition, our method can be generalized to unbounded quantitative properties such as mean-payoff bounds.

In the second part, we introduce two techniques for comparing probabilistic systems. Probabilistic systems are typically compared using the notion of equivalence, which requires the systems to have the equal probability of all behaviors. However, this notion is often too strict, since probabilities are typically only empirically estimated, and any imprecision may break the relation between processes. On the one hand, we propose to replace the Boolean notion of equivalence by a quantitative distance of similarity. For this purpose, we introduce a statistical framework for estimating distances between Markov chains based on their simulation runs, and we investigate which distances can be approximated in our framework. On the other hand, we propose to compare systems with respect to a new qualitative logic, which expresses that behaviors occur with probability one or a positive probability. This qualitative analysis is robust with respect to modeling errors and applicable to many domains.

In the last part, we present a new quantifier-free logic for integer arrays, which allows us to express counting. Counting properties are prevalent in array-manipulating programs, however they cannot be expressed in the quantified fragments of the theory of arrays. We present a decision procedure for our logic, and provide several complexity results.

Acknowledgments

First of all, I want to thank my advisor, prof. Thomas A. Henzinger, for his guidance during my PhD program. I am grateful for the freedom I was given to pursue my research interests, and his continuous support. Working with prof. Henzinger was a truly inspiring experience and taught me what it means to be a scientist.

I want to express my gratitude to my collaborators: Nikola Beneš, Krishnendu Chatterjee, Martin Chmelík, Ashutosh Gupta, Willibald Krenn, Jan Křetínský, Dejan Ničković, Andrey Kupriyanov, and Tatjana Petrov. I have learned a great deal from my collaborators, and without their help this thesis would not be possible. In addition, I want to thank the members of my thesis committee: Dirk Beyer, Dejan Ničković, and Georg Weissenbacher for their advice and reviewing this dissertation. I would especially like to acknowledge the late Helmut Veith, who was a member of my committee. I will remember Helmut for his kindness, enthusiasm, and wit, as well as for being an inspiring scientist.

Finally, I would like to thank my colleagues for making my stay at IST such a pleasant experience: Guy Avni, Sergiy Bogomolov, Ventsislav Chonev, Rasmus Ibsen-Jensen, Mirco Giacobbe, Bernhard Kragl, Hui Kong, Petr Novotný, Jan Otop, Andreas Pavlogiannis, Tantjana Petrov, Arjun Radhakrishna, Jakob Ruess, Thorsten Tarrach, as well as other members of groups Henzinger and Chatterjee.

List of Publications

The author was supported in part by the European Research Council (ERC) under grant 267989 (QUAREM) and by the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE and SHiNE) and Z211-N23 (Wittgenstein Award).

Chapter 3 is an extended version of [1] and is a joint work with Thomas A. Henzinger, Jan Křetínský, and Tatjana Petrov. The theory and discussion of the experiments in this chapter have been expanded. The work in Chapter 4 was published as [2] and is a collaboration with Thomas A. Henzinger, Jan Křetínský, and Tatjana Petrov. Chapter 5 was first published as [3], and later as [4]; this chapter is a joint work with Krishnendu Chatterjee and Martin Chmelík. Chapter 6 is an extended version of [5] and is a joint work with Thomas A. Henzinger and Andrey Kupriyanov. Compared to the original publication, Chapter 6 contains new theoretical results. Chapter 7 contains the work on formal testing published as [6; 7; 8], and is a joint work with Nikola Beneš, Ashutosh Gupta, Thomas A. Henzinger, Willibald Krenn, Jan Křetínský, and Dejan Ničković. Parts of the above-mentioned publications also appear in Chapters 1, 2, and 8.

Table of Contents

1 Introduction

1.1 Motivation

Software systems have become ubiquitous in modern society, with applications ranging from mobile devices to kitchen appliances. We are also witnessing new aspects of our lives being controlled by software, such as autonomous vehicles, smart homes, and wearable technology. At the same time, the complexity of software is growing rapidly, with modern devices having functionality that was difficult to imagine several decades ago.

The high complexity of software systems makes them more likely to contain errors. Thus, as we become more dependent on software, there is an increasing pressure to ensure its correctness. This motivates the field of *formal methods*, which is the science of reasoning about software systems and proving their correctness. Formal methods are based on paradigms such as model checking [9; 10], abstract interpretation [11; 12], and theorem proving [13], which leverage mathematical techniques to analyze programs. Formal methods seek to establish a mathematical proof that a program has a given property on every input, e.g. that the program never throws an exception. This is in contrast to testing, which demonstrates how the program behaves only on the tested inputs, thus cannot validate universal program properties.

Software systems often interact with the physical environment, such as hardware and sensors, which affects their behavior. Consequently, the verification of such systems requires to model physical aspects of the environment. This has motivated researchers to study formal methods in the context of phenomena such as uncertainty, time, and continuous behavior. As a result, formal methods have gone beyond the study of computer programs, and have been applied to the analysis of diverse domains, such as cyber-physical systems [14], stochastic systems [15],

games [16], and biological reaction networks [17].

The grand challenge of formal methods is to develop techniques that automatically verify whether a system satisfies a correctness criterion. There is a great need for automated verification, since systems are typically too complex to analyze their correctness manually. While many verification approaches have been considered in the literature, the following three methods are the most relevant to this thesis:

- *Model checking* is a verification method that aims to check whether a model satisfies a property [9; 10]. The property typically describes the temporal behavior of the system, and is specified in a modal logic such as linear temporal logic (LTL) [18]. Model checking has been applied to the verification of various formalism, including probabilistic systems [15] and games [16].

- In formal methods systems are usually compared using the notions of *equivalence* and *refinement* [19; 20; 21]. Equivalence requires two systems to exhibit the same behaviors, while refinement means that one system has at least as many behaviors as the other. On the one hand, checking equivalence is useful when we wish to know whether one component can be replaced by another component in a system without changing the overall behavior. On the other hand, refinement allows us to certify that an implementation conforms to the system specification.

- *Symbolic methods* aim to represent large or infinite state spaces by logic formulas. Among many application, symbolic methods have been used for the efficient model checking of large systems [22], to express program invariants [23; 24], and for symbolic executions of programs [25; 26]. Symbolic methods have driven the research on logical theories and their decision procedures [27; 28].

In this thesis, we consider the verification of two kinds of systems: *probabilistic systems* and *programs with arrays*. Both kinds are of central interest to formal methods, motivated by the abundance of their applications. Probabilistic behavior occurs naturally in a variety of domains, such as engineering, economy, and biology. Also in software systems probability arises in the form of randomization, or from the interaction with the physical environment in cyber-physical systems. Similarly, arrays are one of the most fundamental data structures, therefore proving

properties of program with arrays is of great importance to formal methods. The analyses of probabilistic systems and programs with arrays are established topics, but this thesis contributes new results to these fields.

At the first glance probabilistic systems and program with arrays seem to have nothing in common. At closer look, however, one can notice that both types of systems extend more classical models by *unbounded behavior*. Formal methods have often focused on the models of computation that are bounded in some aspects. For instance, the classical model checking algorithms reason about Kripke structures with a finite state space. Another example is program analysis, which is often limited to programs with numeric data types only and cannot handle data structures. Data structures, such as arrays, linked lists, and trees, are a convenient method to store unbounded amount of data in a structured way. In theory, a single numeric variable is enough to store any amount of data in a program, but data structures are much more natural for this purpose. Thus, program without data structures in practice can store only limited amount of data in their numeric variables. Probabilistic systems and programs with arrays extend these classical models by unbounded behavior. Probabilistic systems include unbounded behavior in the form of a probability distribution, which describes their state. Thus, the state space of (finite) probabilistic systems is infinite. Program with arrays extend the classical model of programs by making it possible to store unlimited amount of data in arrays.

In Sections 1.2 and 1.3 we introduce the verification techniques for probabilistic systems and program with arrays that constitute the core part of this thesis. Apart from these two key topics, this dissertation contains a briefly summary of the author's work on formal testing. Section 1.4 gives an introduction to formal testing.

1.2 Probabilistic Systems

We consider the analysis of two types of probabilistic systems: *Markov chains (MCs)* and *Markov decision processes (MDPs)*. Markov chains are a popular formalism for modeling systems with the *Markov property*, which means that the probability of making a transition from one state to another depends only on the current state, rather than the states visited previously. Markov decision processes extend Markov chains by admitting non-deterministic choices in addition to probabilistic behavior. MCs and MDPs have been widely adopted as models for randomized

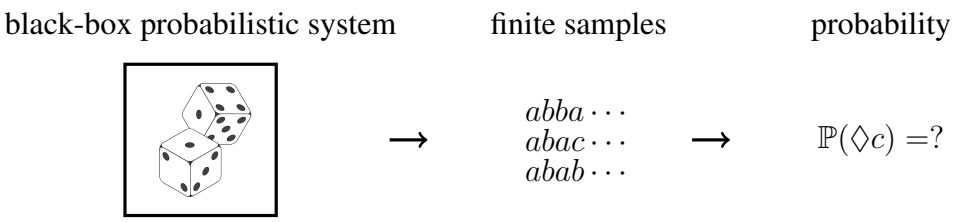

Figure 1.1: Statistical model checking.

protocols, biological networks, task planning, as well as for other domains.

Statistical model checking The traditional approach to the analysis of probabilistic systems is based on numerical algorithms. Numerical algorithms represent a verification problem as a linear program, which is then solved by algebraic methods or iterativly approximated [29]. The drawback of the numerical algorithms is that they require full and precise knowledge of the probabilities in the system. In practice, however, the probability distribution is often unknown or imprecise. For example, probabilities of failures of hardware components are typically only estimated.

This problem motivates the approach of *statistical model checking (SMC)*, where finite paths are sampled from a probabilistic system and statistical estimation is applied to infer conclusion about the system (see Figure 1.1). Thus, statistical model checking requires only to generate simulations of the system from the initial configuration (the *black-box* setting), which is in contrast with the numerical algorithms that require full knowledge of the system (the *white-box* setting).

Statistical model checking provides results in the form of statistical guarantees. This means that answers have a bounded precision (e.g. "the termination probability is in the range $[0.8, 0.9]$") and confidence (e.g. "the result is valid with 99% confidence."). A characteristic feature of SMC is that the number of samples required to infer the conclusion depends mostly on the desired precision and confidence, rather that the size of the system. On the contrary, the numerical algorithms provide a precise and certain solution, but the complexity of the procedure grows with the size of the system. This feature makes SMC an appealing approach also when dealing with systems that are fully known, but where the state space is intractable for the numerical methods.

SMC for unbounded properties Most research in statistical model checking have focused on the verification of properties with a *bounded horizon*, e.g. "a goal state is reached with probability at least 0.5 in the first 1000 steps." Bounded properties have a simple sampling procedure, since it is enough to generate samples bounded by the given horizon. In contrast, SMC of *unbounded properties* (e.g. "a goal state is reached with probability at least 0.5 in *any* number of steps") is more challenging, as it requires a stopping criterion for sample generation. In general, such stopping criterion is impossible to obtain in a completely black-box setting, because we cannot be certain whether a sample could still satisfy the property if extended sufficiently long. As a consequence, all SMC algorithms for unbounded properties require additional knowledge of the system. The previous SMC approaches require relatively detailed information, such as: (i) the system's topology [30], (ii) the second eigenvalue of the Markov chain [30], or (iii) both the size of the Markov chain and the minimum transition probability [31]. Such detailed information may be difficult to obtain, when little is known about the Markov chain.

In Chapter 3 we present a new SMC algorithm for Markov chains with respect to unbounded reachability that provides strong guarantees in the form of confidence bounds. The main idea is to monitor each simulation run on the fly, in order to detect with high probability if the system reached a set of states, called a *bottom strongly connected component*, where the simulation becomes trapped. Once a bottom strongly connected component is entered, the simulation run can be terminated early. As a result, our simulation runs are often much shorter than required by termination bounds that are computed a priori for a desired level of confidence on a large state space. In comparison to the previous algorithms for SMC our method is not only faster in many cases, but also requires less information about the system, namely, only the minimum transition probability that occurs in the Markov chain.

The previous approaches to SMC can handle only the simplest forms of unbounded properties. Typically, they are restricted to reachability properties or the "until" operator (without nesting) which is only a slight generalization of reachability [30]. As a consequence, these approaches cannot handle unbounded properties that are common in verification, but are more general than reachability, such as recurrence ("a behavior repeats infinitely often") or persistence ("eventually a behavior always repeats"). Some researchers did consider richer logics for unbounded properties, however their approaches either require strong assumptions about the Markov chains (e.g. ergodic Markov chains [32], Markov chains without loops [33]), or they provide weak guarantees

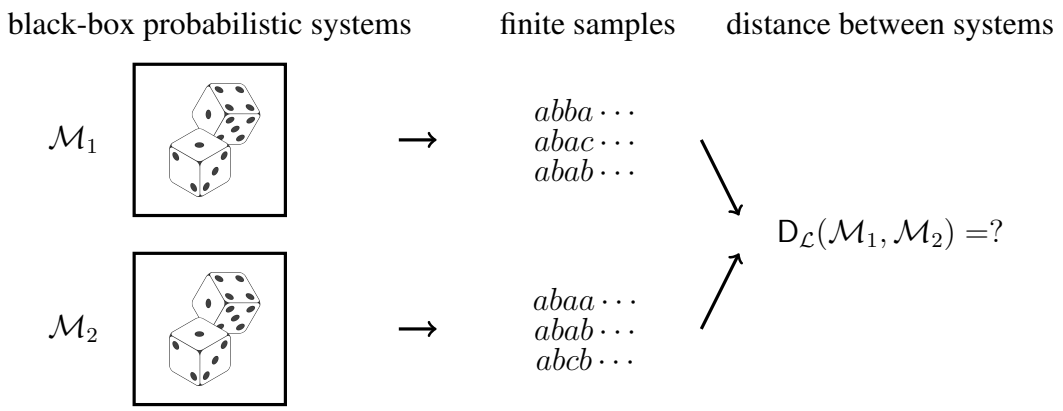

Figure 1.2: Distances between Markov chains.

(e.g. only possible to check if the property holds with a positive probability [34]). Thus, none of the previous SMC approaches can handle a rich temporal logic in a general setting.

We present two extensions to our algorithm that can handle complex unbounded properties in a general setting. First, we present the first SMC algorithm for *linear temporal logic* (LTL). LTL is a rich modal logic that is strictly more expressive than reachability, and which can express recurrence and persistence properties. In addition, we generalize our method to unbounded quantitative properties such as *mean-payoff* bounds. Intuitively, the mean payoff describes the expected average reward per step when the running time goes to infinity. These extensions leverage the fact that checking linear temporal logic as well as mean payoff can be reduced to the analysis of bottom strongly connected components in Markov chains.

Equivalence and refinement of probabilistic systems Probabilistic systems are typically compared using the quantitative notions of equivalence and refinement, which require the systems to exhibit behaviors with the same probability. These notions, however, are often too strict for probabilistic systems, where probabilities are empirically estimated and even the smallest imprecision breaks the equivalence between systems. We provide two solutions to this problems:

- in Chapter 4 we lift the Boolean notion of equivalence to the quantitative *measure of distance between systems*,
- in Chapter 5 we investigate refinement of systems with respect to *qualitative properties*.

Distances between Markov chains In Chapter 4 we consider the problem of measuring a *distance* between two Markov chains by observing only simulation runs, i.e. in the black-box setting. A distance provides a quantitative measure of similarity between two Markov chains, and can range from zero, which indicates full equivalence, to the maximum distance, meaning that the systems are as different as possible. A measure of similarity can help us to choose the component that fits best the desired behavior among several options. Distances are especially useful in the black-box setting, where probabilities can only be approximated, and thus the quantitative notion of equivalence, which requires equal probabilities, cannot be decided.

We investigate distances based on linear behavior, which are the most appropriate when dealing with simulation runs. Among linear distances, we focus our attention on the *total variation distance* and *finite trace distance*. The total variation distance is the maximum difference in probabilities of the two systems among all events. In contrast, the finite trace distance describes the difference in probabilities among all finite traces. In addition, we consider several other distances given topologically, as well as by temporal logics and automata.

Our key idea is to introduce a *framework for measuring distances between Markov chains in a black-box setting*. The input to the framework are Markov chains that can be simulated on demand, and the result is a confidence interval for a given distance (see Figure 1.2). We investigate which distances can be estimated in our framework, and we provide both negative and positive results. The main negative result is that the total variation distance cannot be estimated. Intuitively, this is caused by the non-robustness of the total variation distance, which means that even the smallest difference between the Markov chains can maximize this distance. In contrast, our main positive result is a method for estimating the finite trace distance; in addition we provide methods for estimating several distances given topologically, by temporal logics, and automata.

Qualitative analysis of probabilistic system In Chapter 5 we aim to compare probabilistic systems with respect to *qualitative properties*. Qualitative properties express that desired behaviors of the system arise almost-surely (with probability one) or with a positive probability. Qualitative analysis of systems is robust with respect to modeling errors, thus small imprecision does not break refinement between processes. Moreover, there are many applications, where we need to know whether the correct behavior arises with probability one. For instance, when ana-

lyzing a randomized embedded scheduler, we are interested in whether every thread progresses with probability one.

We consider qualitative analyses of Markov decision processes that are fully known, i.e. in the white-box setting. Our key problem is to decide whether one MDP refines another MDP with respect to qualitative properties. The notion of refinement is especially useful, when one MDP is treated as a probabilistic specification, and the other as an implementation. To this end, we describe a new logic QCTL^* that can expresses qualitative properties of MDPs. Furthermore, we introduce a relation between MDPs, called *combined simulation*, that captures refinement with respect to QCTL^*. Finally, we present an algorithm with quadratic complexity to compute combined simulation.

Markov decision processes are frequently used for modeling of concurrent probabilistic systems, because non-determinism models context switching in a natural way. With this in mind, we study the problem of computing the combined simulation for a concurrent system that consists of multiple MDPs. The naive solution is to construct an explicit parallel composition of the MDPs and then compute the simulation relation. This approach, however, does not scale since parallel composition of two MDPs may result in a system of quadratic size. This is an example of the *state explosion* problem in parallel systems, where the size of the state space may grow exponentially in the number of components.

To tackle the state explosion problem, we propose an *assume-guarantee* approach to computing the combined simulation. Roughly speaking, assume-guarantee is a style of reasoning about concurrent systems, where some processes are replaced by their abstraction. The advantage of the assume-guarantee approach is that the composition with an abstraction may be much smaller than the direct composition of processes. We provide an assume-guarantee algorithm for checking the combined simulation, which follows the approach of *counterexample-guided abstraction refinement (CEGAR)* [35]. In our algorithm an abstraction is iterativly refined until the simulation can be established or a valid counterexample to the simulation is found. We illustrate on several well-known examples that our assume-guarantee algorithm can outperform the naive approach.

Interestingly, we established our results by showing a tight link between two-player games and MDPs. We show that for the sake of qualitative analysis, an MDP can be interpreted as a game between the probability and the scheduler, which resolves the non-determinism. As a

consequence, our results on refinement and assume-guarantee analysis of MDPs are lifted to games.

1.3 Programs with Arrays

Arrays are prevalent in programs, for instance every C and Java program uses an array to pass command-line arguments. Many popular data structures, such as hash tables and trees, are implemented on top of arrays. The importance of arrays has motivated researchers to develop logics that can express array properties. Such logics can be used both in the Floyd–Hoare style to specify program invariants in order to prove program safety [23], and in symbolic execution in order to test programs [25; 36]. Logics for arrays extend other theories by array variables, as well as by array read and write operations. An example of a decidable extension is the quantifier-free Presburger arithmetic with arrays.

Expressing non-trivial properties of arrays, such as sortedness, requires universal quantification over array indexes. Unfortunately, many quantified theories of arrays are undecidable, e.g. quantified Presburger arithmetic with arrays. To sidestep the undecidability of the general theories of arrays, researchers have focused on restricted forms of universal quantification [24; 37; 38; 39]. Logics with restricted quantification can express many practical properties of arrays, e.g. sortedness or that all array elements belong to a bounded range. However, an important class of properties, namely *counting over arrays*, cannot be expressed in the previous decidable logics for arrays. Counting properties, such as "there are equally many zeros and ones in the array" and "the length of fields in an IP packet is correct" occur frequently in programs, and thus pose an interesting verification problem.

In Chapter 6 we present *array folds logic* (AFL), a new quantifier-free logic that can express counting properties of arrays. Our logic is motivated by the observation that many counting properties follow the *fold pattern*, which is a concept well-known from functional languages. Roughly speaking, folding works by iterating over an array and evaluating a function on every element using the values computed from the previous elements. We introduce this concept into AFL in the form of the *fold term*, which folds a function over an array. We consider a special type of functions, which utilize counters to accumulate values while traversing an array. In essence, the fold term replaces universal quantification as a method to express universal properties of

```
zeros = 0; ones = 0;
for(i=0;i<size(a);i++) {
  if(a[i]==0) zeros++;
  if(a[i]==1) ones++;
}
assert(zeros == ones);
```

(a) C language.

$$fold_a \begin{pmatrix} \mathbf{i}{=}0 \\ \mathbf{c}_0{=}0 \\ \mathbf{c}_1{=}0 \end{pmatrix} \begin{pmatrix} \mathbf{e}{=}0 \Rightarrow \mathbf{c}_0{++} \\ \mathbf{e}{=}1 \Rightarrow \mathbf{c}_1{++} \\ \mathbf{e}{\neq}0 \wedge \mathbf{e}{\neq}1 \Rightarrow \mathit{skip} \end{pmatrix} = \begin{pmatrix} |a| \\ z \\ o \end{pmatrix}$$

$$\wedge\, z = o$$

(b) Array folds logic.

Figure 1.3: Program that requires equal number of zeros and ones in an array, and its AFL representation.

arrays.

Figure 1.3 illustrates the idea of AFL. The program in Figure 1.3a uses the variables `zeros` and `ones` to count the number of zeros and ones in an array, and the assertion at the last line requires these counts to match. Figure 1.3b shows how this constraint can be expressed as an AFL formula. The fold term in the formula uses two counters: $\mathbf{c}_0$ and $\mathbf{c}_1$ to count the number of zeros and ones, respectively. The expression $\begin{pmatrix} \mathbf{i}{=}0 \\ \mathbf{c}_0{=}0 \\ \mathbf{c}_1{=}0 \end{pmatrix}$ specifies the initial values for folding: that folding starts at the beginning of the array and the counters are initialized to zero. The expression in the following bracket specifies the fold function, which is consecutively applied to every array element. The variable $\mathbf{e}$ in the fold function denotes the value of the element being inspected. The top two branches of the fold function specify that if the element value equals zero or one, then the respective counter is incremented. In case the element has a different value, the bottom branch orders the function to skip. The fold term returns the vector of three values: the length of the array $|a|$, the final value of $\mathbf{c}_0$ in the variable z, and the final value of $\mathbf{c}_1$ in the variable o. The last part of the AFL formula requires the variables z and o to be equal.

We provide several interesting results about the complexity of AFL. We have proved that AFL has the *small model property*, meaning that if a formula is satisfiable, then it has a model that can be encoded using space polynomial in the size of the formula. Based on this result, we prove that the satisfaction problem of AFL is **PSPACE**-complete. We also show that with a natural restriction the complexity decreases to **NP**. Finally, we prove that several natural extensions of AFL lead to undecidability.

Apart from the complexity analysis, we present a decision procedure for AFL, which

translates an AFL formula to a quantifier-free Presburger formula. This type of Presburger formulas can be handled by efficient solvers, which makes our decision procedure practical. In addition, we provide a method for generating a satisfying assignment for AFL formulas. We demonstrate on practical examples that our decision procedure can solve a broad range of problems in symbolic testing and program verification. The example in Figure 1.3 illustrates another feature of AFL, namely the ability to concisely summarize loops with internal branching that perform counting. To the best of our knowledge, this class of loops cannot be handled by the previous approaches to loop summarization.

1.4 Formal Testing

In Chapter 7 we shortly summarize the author's work on formal testing. This section provides an introduction to the work discussed in that chapter.

Testing — to quote Dijkstra — can only show the presence of bugs, but not their absence. Despite this limitation, testing remains the preferred method for quality assurance in software engineering. Testing remains popular because it scales to very large systems, and it is relatively easy to setup. In addition, testing can find bugs directly in the executable, rather than the program specification like verification methods. Another advantage of testing is that it produces only sound bug reports, which is convenient for debugging. On the contrary, many verification techniques produce false alarms. Finally, testing can be easily applied to data-driven applications, such as signal processing, while formal methods have been mostly studied in the context of small control-driven applications, such as device drivers.

Formal testing is an approach to program analysis that combines testing and formal methods. The goal of formal testing is to combine the best of both worlds: the high coverage offered by verification, and the scalability of testing. Methods that combine both approaches can outperform both verification tools at proving correctness, and testing tools at finding bugs [40; 41; 42]. By augmenting testing with formal methods we can achieve better guarantees and target corner cases that manually-created test cases often miss. Furthermore, formal methods can improve the testing process by providing the methods to analyze and shape test specifications.

Model-based testing *Model-based testing* is an approach to formal testing that checks whether an implementation, given as a black box, conforms to a formal specification. The specification describes the desired behavior of the system, and is derived from the system's requirements. The specification is used to automatically generate test cases that can discriminate whether the implementation conforms to the specification. Our work [6; 7] on model-based testing revolves around the **ioco** framework, where specification are transitions systems with input and output actions [43].

Software engineering often deals with systems that consists of multiple components. Such systems require integration testing, i.e. checking that components work correctly when combined into a system. Integration testing can be a laborious task and, even worse, is often repeated each time a component is modified. In Chapter 7 we discuss our work on *compositional properties* of the **ioco** theory. Compositional properties make it possible to decrease the amount of integration testing by reusing testing information that we have already gained by testing the individual components, and the previous versions of the system. For this purpose, we study the compositional operators, which allow us to identify the part of the system that is affect by the changes to the components. Consequently, we can avoid the integration testing of the entire systems when a single component is modified, and instead retest only the affected part.

Abstraction-driven concolic testing The quality of software tests is typically measured by a *coverage criterion*. Roughly speaking, a coverage criterion is a collection of program elements that we wish to reach. For example, the goal of the branch coverage is to execute as many program branches as possible. Test coverage is measured by the ratio of the exercised elements to the total number of elements. Certification bodies require a test suite that achieves high coverage test as an evidence of software quality, e.g. the DO-178C standard in avionics.

Concolic testing is a popular method for test case generation [36]. This method is essentially symbolic execution augmented by concrete execution, such that symbolic expressions that fall outside of a decidable theory are simplified by concrete values. Concolic testing suffers from the path explosion problem, so it often fails to reach coverage goals that are deep in the program. Several heuristics for path exploration have been proposed that try to maximize the coverage of concolic testing [44; 45; 46], e.g., randomly picking program branches to explore, driving exploration toward uncovered branches that are closest to the last explored branch, etc. These

heuristics, however, are limited by their local view of the program semantics, i.e. they are only aware of the (in)feasibility of the paths seen so far.

In contrast to testing, *software model checkers* compute program abstractions that contain all feasible program paths [47]. Due to abstraction, not all paths contained in the abstraction are guaranteed to be feasible, therefore abstract model checking is not directly useful for generating test suites. In Chapter 7 we present a new algorithm to guide concolic testing by a model checker [8]. The algorithm iterativly combines concolic testing and model checking, such that concolic testing is guided by a program abstraction and the abstraction is refined for the remaining test goals.

2 Preliminaries

In this chapter we recall some basic concepts of formal methods that are used in this thesis. Section 2.1 covers discrete systems, such as automata and games, as well as modal logics for expressing their properties. Section 2.2 recalls basic concepts of probability theory and Markov processes. Finally, Section 2.3 explains techniques of statistical inference.

2.1 Discrete Systems

2.1.1 Finite and Infinite Words

Let Ap denote a non-empty finite set of atomic propositions. Given a finite set S we denote by S^* (respectively S^ω) the set of finite (resp. infinite) sequences of elements from S, and let $S^+ = S^* \setminus \{\epsilon\}$, where ϵ is the empty string.

An ω-*word* is an infinite sequence $\overline{w} = A_0 A_1 \cdots \in (2^{Ap})^\omega$. The i-th letter of $\overline{w}$ is denoted by $\overline{w}[i]$, i.e. $\overline{w}[i] = A_i$, and we write $\overline{w}_i$ for the suffix $\overline{w}[i]\overline{w}[i+1]\cdots$. A *word* w is a finite sequence of symbols $w \in (2^{Ap})^*$. We define $w[i]$ and w_i for words in a similar way as for ω-words.

2.1.2 Linear Temporal Logic

Linear temporal logic (LTL) is a modal logic for expressing temporal behaviors of system [48]. The formulas of LTL are given by the following syntax in the positive normal form [10]:

$$\varphi \ ::= ap \mid \neg ap \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi\mathcal{U}\varphi \mid \varphi\mathcal{W}\varphi$$

for $ap \in Ap$.

The semantics of LTL is defined with respect to an ω-word $\overline{w} \in (2^{Ap})^\omega$:

$$
\begin{array}{lll}
\overline{w} \models ap & \text{iff} & a \in \overline{w}[0] \\
\overline{w} \models \neg ap & \text{iff} & a \notin \overline{w}[0] \\
\overline{w} \models \varphi_1 \vee \varphi_2 & \text{iff} & \overline{w} \models \varphi_1 \text{ or } \overline{w} \models \varphi_2 \\
\overline{w} \models \varphi_1 \wedge \varphi_2 & \text{iff} & \overline{w} \models \varphi_1 \text{ and } \overline{w} \models \varphi_2 \\
\overline{w} \models \bigcirc\varphi & \text{iff} & \overline{w}_1 \models \varphi \\
\overline{w} \models \varphi_1 \mathcal{U} \varphi_2 & \text{iff} & \exists\, j \in \mathbb{N} : (\overline{w}_j \models \varphi_2 \text{ and } \forall\, 0 \leq i < j : \overline{w}_i \models \varphi_1) \\
\overline{w} \models \varphi_1 \mathcal{W} \varphi_2 & \text{iff} & \varphi_1 \mathcal{U} \varphi_2 \text{ or } \forall j \in \mathbb{N} : \overline{w}_j \models \varphi_1.
\end{array}
$$

We write $\mathcal{L}(\varphi)$ for the the set of ω-words that satisfy the formula φ:

$$\mathcal{L}(\varphi) \;=\; \{\overline{w} \in (2^{Ap})^\omega \mid \overline{w} \models \varphi\}.$$

We use the following shorthands for LTL formulas:

$$
\begin{array}{lll}
\mathsf{true} & \equiv & q \vee \neg q \;\text{ for some } q \in Ap \\
\mathsf{false} & \equiv & q \wedge \neg q \;\text{ for some } q \in Ap \\
\Diamond\varphi & \equiv & \mathsf{true}\ \mathcal{U}\ \varphi \\
\Box\varphi & \equiv & \varphi\ \mathcal{W}\ \mathsf{false}.
\end{array}
$$

2.1.3 Deterministic Rabin Automata

A deterministic Rabin automaton is a model of computation that accepts infinite strings of symbols, i.e. ω-words. This type of automata is important for LTL model checking, since for every LTL formula φ, one can construct a deterministic Rabin automaton that accepts all ω-words that satisfy φ.

Definition 2.1 (Deterministic Rabin automaton)**.** *A* deterministic Rabin automaton *(DRA) is a tuple* $\mathcal{A} = (Q, \gamma, q_o, Acc)$*, where*

- Q *is a finite set of states,*
- $\gamma : Q \times 2^{Ap} \to Q$ *is a transition function,*
- $q_o \in Q$ *is an initial state,*
- $Acc \subseteq 2^Q \times 2^Q$ *is an acceptance condition.*

An ω-word $\overline{w}$ induces in a DRA $\mathcal{A}$ an infinite sequence of states $\mathcal{A}(\overline{w}) = s_0 s_1 \cdots \in Q^\omega$, such that $s_0 = q_o$ and $\gamma(s_i, \overline{w}[i]) = s_{i+1}$ for $i \geq 0$. We write $\mathrm{Inf}(\overline{w})$ for the set of states that occur infinitely often in $\mathcal{A}(\overline{w})$. Word $\overline{w}$ is *accepted*, if there exists a pair $(E, F) \in Acc$, such that $E \cap \mathrm{Inf}(\overline{w}) = \emptyset$ and $F \cap \mathrm{Inf}(\overline{w}) \neq \emptyset$. The language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of all words accepted by $\mathcal{A}$. The class of languages accepted by DRAs is known as *ω-regular languages*. The following is a well-known result [10]:

Lemma 2.1. *For every* LTL *formula φ, a DRA $\mathcal{A}_\varphi$ can be effectively constructed such that $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$.*

2.1.4 Two-Player Games

Games are useful for modeling systems with several agents, who may have conflicting objectives. In this thesis we consider games with two players.

Definition 2.2 (Two-player game)**.** *A* two-player game *is a tuple $G = (S, A, \mathsf{Av}, \delta, L, s_0)$, where*

- *S is a finite set of states,*
- *A is a finite set of actions,*
- $\mathsf{Av} : S \to 2^A \setminus \emptyset$ *is an* action-available *function that assigns to every state $s \in S$ the set $\mathsf{Av}(s)$ of actions available in s,*
- $\delta : S \times A \to 2^S \setminus \emptyset$ *is a non-deterministic* transition *function that given a state $s \in S$ and an action $a \in \mathsf{Av}(s)$ gives the set $\delta(s, a)$ of successors of s given action a,*
- $L : S \to 2^{Ap}$ *is a* labeling *function,*
- *$s_0 \in S$ is an initial state.*

Plays A two-player game is played for infinitely many rounds as follows: the game starts in the initial state, and in every round Player 1 chooses an available action from the current state and then Player 2 chooses a successor state, and the game proceeds to the successor state for the next round. Formally, a *play* in a two-player game is an infinite sequence $\rho = s_0 a_0 s_1 a_1 s_2 a_2 \cdots$ of states and actions such for all $i \geq 0$ we have that $a_i \in \mathsf{Av}(s_i)$ and $s_{i+1} \in \delta(s_i, a_i)$. We denote by Ω the set of all plays.

Turn-based games A two-player game G is *turn-based* if in every state either Player 1 or Player 2 can make choices. Formally, for all $s \in S$ we have either (i) $|\mathsf{Av}(s)| = 1$ (then we refer to s as a Player-2 state), or (ii) for all $a \in \mathsf{Av}(s)$ we have $|\delta(s, a)| = 1$ (then we refer to s as a Player-1 state).

Strategies Strategies are recipes that describe how to extend finite prefixes of plays. Formally, a *strategy* for Player 1 is a function $\sigma : (S \times A)^* \times S \to A$, which given a finite history $w \cdot s \in (S \times A)^* \times S$ of the game gives an action from $\mathsf{Av}(s)$ to be played next. We write Σ for the set of all Player-1 strategies. A strategy for Player 2 is a function $\theta : (S \times A)^+ \to S$, which given a finite history $w \cdot s \cdot a$ of a play selects a successor state from the set $\delta(s, a)$. We write Θ for the set of all Player-2 strategies. *Memoryless* strategies are independent of the history, but depend only on the current state for Player 1 (resp. the current state and action for Player 2) and hence can be represented as functions $S \to A$ for Player 1 (resp. as functions $S \times A \to S$ for Player 2).

Outcomes Given a strategy σ for Player 1 and θ for Player 2 the *outcome* is a unique play, denoted as $\mathsf{Play}(s, \sigma, \theta) = s_0 a_0 s_1 a_1 \cdots$, which is defined as follows: (i) $s_0 = s$, and (ii) for all $i \geq 0$ we have $a_i = \sigma(s_0 a_0 \ldots s_i)$ and $s_{i+1} = \theta(s_0 a_0 \ldots s_i a_i)$. Given a state $s \in S$ we denote by $\mathsf{Plays}(s, \sigma)$ (resp. $\mathsf{Plays}(s, \theta)$) the set of possible plays given σ (resp. θ), i.e., $\bigcup_{\theta' \in \Theta} \mathsf{Play}(s, \sigma, \theta')$ (resp. $\bigcup_{\sigma' \in \Sigma} \mathsf{Play}(s, \sigma', \theta)$).

2.1.5 Alternating-time Temporal Logic

We consider *alternating-time temporal logic* (ATL^*) [16] as a logic to specify properties for two-player games. ATL^* is a generalization of linear temporal logic introduced in Section 2.1.2. ATL^* allows us to specify whether players collaborate to achieve an objective ($\langle\langle 1, 2 \rangle\rangle$ quantifier), collaborate to avoid the objective ($\langle\langle \emptyset \rangle\rangle$), or play against each other ($\langle\langle 1 \rangle\rangle$ and $\langle\langle 2 \rangle\rangle$ quantifiers).

The syntax of ATL^* is given in the positive normal form by defining the set of *path formulas* (φ) and *state formulas* (ψ) according to the following grammar:

$$\begin{array}{lll} \text{state formulas:} & \psi & ::= ap \mid \neg ap \mid \psi \vee \psi \mid \psi \wedge \psi \mid \mathrm{PQ}(\varphi) \\ \text{path formulas:} & \varphi & ::= \psi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \bigcirc \varphi \mid \varphi \mathcal{U} \varphi \mid \varphi \mathcal{W} \varphi, \end{array}$$

where $ap \in Ap$ is an atomic proposition and PQ is a path quantifier. The path quantifiers PQ are as follows:

$$\mathrm{ATL}^* \text{ path quantifiers: } \langle\langle 1 \rangle\rangle, \langle\langle 2 \rangle\rangle, \langle\langle 1, 2 \rangle\rangle, \text{ and } \langle\langle \emptyset \rangle\rangle.$$

Given a play $\rho = s_0 a_0 s_1 a_1 \cdots$ we denote by $\rho[i]$ the i-th state of ρ, i.e. $\rho[i] = s_i$ and we write ρ_i for the suffix starting at the i-th state element of the play ρ, i.e., $\rho_i = s_i a_i s_{i+1} a_{i+1} \cdots$. The semantics of path formulas is defined inductively as follows:

$$\begin{array}{lll}
\rho \models \psi & \text{iff} & \rho[0] \models \psi \\
\rho \models \varphi_1 \vee \varphi_2 & \text{iff} & \rho \models \varphi_1 \text{ or } \rho \models \varphi_2 \\
\rho \models \varphi_1 \wedge \varphi_2 & \text{iff} & \rho \models \varphi_1 \text{ and } \rho \models \varphi_2 \\
\rho \models \bigcirc \varphi & \text{iff} & \rho_1 \models \varphi \\
\rho \models \varphi_1 \mathcal{U} \varphi_2 & \text{iff} & \exists j \in \mathbb{N} : (\rho_j \models \varphi_2 \text{ and } \forall 0 \leq i < j : \rho_i \models \varphi_1) \\
\rho \models \varphi_1 \mathcal{W} \varphi_2 & \text{iff} & \rho \models \varphi_1 \mathcal{U} \varphi_2 \text{ or } \forall j \in \mathbb{N} : \rho_j \models \varphi_1.
\end{array}$$

Given a path formula φ and game G, we denote by $[\![\varphi]\!]_G$ the set of plays ρ in G such that $\rho \models \varphi$. We omit the G lower script when the game is clear from context. The semantics of state formulas for ATL^* is defined as follows:

$$\begin{array}{lll}
s \models ap & \text{iff} & ap \in L(s) \\
s \models \neg ap & \text{iff} & ap \notin L(s) \\
s \models \psi_1 \vee \psi_2 & \text{iff} & s \models \psi_1 \text{ or } s \models \psi_2 \\
s \models \psi_1 \wedge \psi_2 & \text{iff} & s \models \psi_1 \text{ and } s \models \psi_2 \\
s \models \langle\langle 1 \rangle\rangle(\varphi) & \text{iff} & \exists \sigma \in \Sigma, \forall \theta \in \Theta : \mathsf{Play}(s, \sigma, \theta) \in [\![\varphi]\!] \\
s \models \langle\langle 2 \rangle\rangle(\varphi) & \text{iff} & \exists \theta \in \Theta, \forall \sigma \in \Sigma : \mathsf{Play}(s, \sigma, \theta) \in [\![\varphi]\!] \\
s \models \langle\langle 1, 2 \rangle\rangle(\varphi) & \text{iff} & \exists \sigma \in \Sigma, \exists \theta \in \Theta : \mathsf{Play}(s, \sigma, \theta) \in [\![\varphi]\!] \\
s \models \langle\langle \emptyset \rangle\rangle(\varphi) & \text{iff} & \forall \sigma \in \Sigma, \forall \theta \in \Theta : \mathsf{Play}(s, \sigma, \theta) \in [\![\varphi]\!],
\end{array}$$

where $s \in S$ and $ap \in Ap$. Given an ATL^* state formula ψ and a two-player game G, we denote by $[\![\psi]\!]_G = \{s \in S \mid s \models \psi\}$ the set of states that satisfy the formula ψ. We omit the G lower script when the game is clear from context.

2.2 Probabilistic Systems

2.2.1 Probability Space and Random Variables

We recall the basic notions from the probability theory. Given a set $\mathcal{O}$ of outcomes, a σ-algebra is a set of subsets of $\mathcal{O}$ that contains the empty set, and is closed under complement and finite union.

Definition 2.3 (σ-algebra)**.** *A σ-algebra of a non-empty set $\mathcal{O}$ is a set $\mathcal{F} \subseteq 2^{\mathcal{O}}$ such that:*

- $\emptyset \in \mathcal{F}$,
- *if $\mathcal{E} \in \mathcal{F}$, then $\mathcal{O} \setminus \mathcal{E} \in \mathcal{F}$,*
- *if $\mathcal{E}_1, \mathcal{E}_2, \cdots \in \mathcal{F}$, then $\bigcup_{i \geq 1} \mathcal{E}_i \in \mathcal{F}$.*

A probability space is a σ-algebra equipped with a probability measure $\mathbb{P}$.

Definition 2.4 (Probability space)**.** *A probability space is a triple $(\mathcal{O}, \mathcal{F}, \mathbb{P})$, where*

- *$\mathcal{F}$ is a σ-algebra of $\mathcal{O}$,*
- *$\mathbb{P} : \mathcal{F} \to [0, 1]$ is a probability measure, such that (i) $\mathbb{P}(\mathcal{O}) = 1$, and (ii) if $\mathcal{E}_1, \mathcal{E}_2, \cdots$ are disjoint, then $\mathbb{P}(\bigcup_{i \leq 1} \mathcal{E}_i) = \sum_{i \leq n} \mathbb{P}(\mathcal{E}_i)$.*

The elements of $\mathcal{F}$ are called *events* and are said to be *measurable*.

Random variables and cumulative distribution function A *random variable* is a function $X : \mathcal{O} \to \mathbb{R}$ that is measurable, i.e. for every $x \in \mathbb{R}$ the set $\{y \in \mathcal{O} \mid X(y) \leq x\}$ belongs to $\mathcal{F}$. A random variable is called *discrete*, if it admits countably many values, and is called *continuous* otherwise. A random variable is called *Bernoulli* if it admits only two possible outcomes: 0 and 1. The *expected value* $\mathbb{E}(X)$, or *mean*, of a random variable X is defined as:

$$\mathbb{E}(X) \;=\; \begin{cases} \sum_x x\mathbb{P}(\{y \in \mathcal{O} \mid X(y) = x\}) & \text{if } X \text{ is discrete} \\ \int_{\mathcal{O}} X \, d\,\mathbb{P} & \text{if } X \text{ is continuous.} \end{cases}$$

The *cumulative distribution function* (CDF) of a random variable X is a function $F_X : \mathbb{R} \to [0, 1]$ defined by

$$F_X(x) \;=\; \mathbb{P}(\{y \in \mathcal{O} \mid X(y) \leq x\}).$$

Independence and conditional probability Given two events $\mathcal{E}$ and $\mathcal{E}'$, we write $\mathbb{P}(\mathcal{E}, \mathcal{E}')$ for $\mathbb{P}(\mathcal{E} \cap \mathcal{E}')$. We say that events $\mathcal{E}$ and $\mathcal{E}'$ are *independent* if $\mathbb{P}(\mathcal{E}, \mathcal{E}') = \mathbb{P}(\mathcal{E})\mathbb{P}(\mathcal{E}')$. Two random variables X and Y are independent if for every sets A and B it holds that the events $A \in X$ and $B \in Y$ are independent. Assuming that $\mathbb{P}(\mathcal{E}') > 0$, the *conditional probability* of $\mathcal{E}$ given $\mathcal{E}'$ is defined as:

$$\mathbb{P}(\mathcal{E} \mid \mathcal{E}') \;=\; \frac{\mathbb{P}(\mathcal{E}, \mathcal{E}')}{\mathbb{P}(\mathcal{E}')}.$$

2.2.2 Markov Chains

Markov chains are random processes that are "memoryless," which means that the probability of going from state to another depends only on the current state. We consider Markov chains that are called discrete-time Markov chains in the literature [10].

Definition 2.5 (Markov chain)**.** *A* Markov chain (MC) *is a tuple* $\mathcal{M} = (S, \mathbf{P}, L, \mu)$*, where*

- S *is a finite set of states,*
- $\mathbf{P} \;:\; S \times S \to [0, 1]$ *is a transition probability matrix, such that for every* $s \in S$ *it holds* $\sum_{s' \in S} \mathbf{P}(s, s') = 1$*,*
- $L : S \to 2^{Ap}$ *is a* labeling *function,*
- $\mu : S \to [0, 1]$ *is an initial distribution, such that* $\sum_{s' \in S} \mu(s, s') = 1$*.*

A *run* of $\mathcal{M}$ is an infinite sequence $\rho = s_0 s_1 \cdots \in S^\omega$ of states, such that for all $i \geq 0$, $\mathbf{P}(s_i, s_{i+1}) > 0$; we let $\rho[i]$ denote the state s_i. We denote by $\mathsf{Runs}_{\mathcal{M}}$ the set of all runs of $\mathcal{M}$. A *path* π in $\mathcal{M}$ is a finite prefix of a run of $\mathcal{M}$. We denote the empty path by λ and concatenation of paths π_1 and π_2 by $\pi_1 \cdot \pi_2$. Each path π in $\mathcal{M}$ determines the set of runs $\mathsf{Cone}(\pi)$ consisting of all runs that start with π:

$$\mathsf{Cone}(s_0 s_1 \cdots s_n) = \{\rho \in \mathsf{Runs}_{\mathcal{M}} \mid \forall 0 \leq i \leq n \;:\; s_i = \rho[i]\}.$$

To $\mathcal{M}$ we assign the probability space $(\mathsf{Runs}_{\mathcal{M}}, \mathcal{F}, \mathbb{P}_{\mathcal{M}})$, where $\mathcal{F}$ is the smallest σ-algebra that contains all cones $\mathsf{Cone}(\pi)$, and $\mathbb{P}_{\mathcal{M}}$ is the unique probability measure such that

$$\mathbb{P}_{\mathcal{M}}(\mathsf{Cone}(s_0 s_1 \cdots s_n)) = \mu(s_0) \prod_{i=1}^{n} \mathbf{P}(s_{i-1}, s_i).$$

where the empty product equals 1. Further, we write $\mathbb{P}_{s,\mathcal{M}}$ for the probability measure, where $\mu(s) = 1$ and $\mu(s') = 0$ for $s' \neq s$. We omit the subscript $\mathcal{M}$ from $\mathbb{P}_{\mathcal{M}}$ and $\mathbb{P}_{s,\mathcal{M}}$ if the Markov chain is clear from the context. The *support* of the probability distribution $\mathbf{P}(s)$ is $\mathrm{Supp}(\mathbf{P}(s)) = \{s' \in S \mid \mathbf{P}(s, s') > 0\}$.

We extend the labeling notation so that for a path $\pi \in S^k$, the projected sequence $L(\pi)$ is the word $w \in \Sigma^k$, where $w[i] = L(\pi[i])$. We overload the notation and for a path π write $\mathbb{P}(\pi)$ meaning $\mathbb{P}(\mathsf{Cone}(\pi))$, and for a ($\omega$)-word w, we write $\mathbb{P}(w)$ meaning $\mathbb{P}(L^{-1}(w))$.

Bottom strongly connected components A Markov chain $\mathcal{M}$ has an underlying directed graph, where the states of $\mathcal{M}$ are vertices, and there is an edge from state s to s' if and only if $\mathrm{Pr}(s, s') > 0$. We use the standard notions of graphs to the underlying graphs of Markov chains.

A non-empty set $C \subseteq S$ of states is *strongly connected* if for every $s, s' \in C$ there is a path from s to s'. A set of states $C \subseteq S$ is a *bottom strongly connected component* (BSCC) of $\mathcal{M}$, if (i) C is strongly connected, and (ii) for each $s \in C$ and $s' \in S \setminus C$ there is no edge from s to s'. The set of all sets that are strongly connected in $\mathcal{M}$ is denoted by SC; similarly the set of all BSCCs in $\mathcal{M}$ is denoted by BSCC.

Note that with probability 1, the set of states that appear infinitely many times on a run forms a BSCC.

Lemma 2.2. *For a finite Markov chain $\mathcal{M}$:*

$$\mathbb{P}(\{\rho \in \mathsf{Runs}_{\mathcal{M}} \mid \mathrm{Inf}(\rho) \cap \mathsf{BSCC}\}) = 1.$$

2.2.3 Markov Reward Models and Mean Payoff

We consider Markov chain with states labeled by a reward function. Rewards are useful for modeling performance of probabilistic systems.

Definition 2.6 (Markov reward model)**.** *A* Markov reward model (MRM) *is a tuple $(\mathcal{M}, r)$, where $\mathcal{M}$ is a Markov chain with state space S, and $r : S \to [0, 1]$ is a reward function.*

Mean payoff (also called long-run average reward) is the average reward per step that we expect to observe, when a Markov runs for a time that goes to infinity. Mean payoff is a good measure of long-time average performance of probabilistic system.

Definition 2.7 (Mean payoff)**.** *The* mean payoff *for a state* $s \in S$

$$\mathsf{MP}(s) \;=\; \lim_{n\to\infty} \mathbb{E}\left(\frac{1}{n}\sum_{i=1}^{n} r(S_{i,s})\right),$$

where $S_{i,s}$ *is a random variable that returns the* i*-th state on a run that starts in the state* s*.*

The mean payoff value always exists [29].

2.2.4 Markov Decision Processes

Markov decision processes are an extension of Markov chains, which admit non-deterministic behavior.

Definition 2.8 (Markov decision process)**.** *A* Markov decision process *(MDP) is a tuple* $\mathcal{M} = (S, (S_1, S_P), A, \mathsf{Av}, \delta_1, \mathbf{P}, L, s_0)$*, where*

- S *is a finite set of states with a partition of* S *into Player-1 states* S_1 *and probabilistic states* S_P*,*
- A *is a finite set of actions,*
- $\mathsf{Av} : S_1 \to 2^A \setminus \emptyset$ *is an action-available function that assigns to every Player-1 state the non-empty set* $\mathsf{Av}(s)$ *of actions available in* s*,*
- $\delta_1 : S_1 \times A \to S$ *is a deterministic transition function that given a Player-1 state and an action gives the next state,*
- $\mathbf{P} : S_P \times S \to [0, 1]$ *is a probabilistic transition function, such that given a probabilistic state* $s \in S_P$ *it holds* $\sum_{s' \in S} \mathbf{P}(s, s') = 1$*,*
- $L : S \to 2^{Ap}$ *is a labeling function,*
- $s_0 \in S$ *is an initial state.*

For technical convenience, we define MDPs with initial states rather than initial distributions as in Markov chains.

$1\frac{1}{2}$-player interpretation Strategies for Player 1 and plays in MDPs are defined as for games. Once a strategy $\sigma \in \Sigma$ for Player 1 is fixed, the outcome of the MDP is a random walk for which the probabilities of events are uniquely defined, where an event $\mathcal{E} \subseteq \Omega$ is a measurable set of plays [49]. For an MDP $\mathcal{M}$, state $s \in S$ and event $\mathcal{E} \subseteq \Omega$, we write $\mathbb{P}^{\sigma}_{s,\mathcal{M}}(\mathcal{E})$ for the probability that a play belongs to $\mathcal{E}$ if the game starts from the state s and Player 1 follows strategy σ. We omit the subscript $\mathcal{M}$ if the MDP is clear from the context.

2.2.5 Probabilistic Computation Tree Logic

In the previous sections, we have seen logics LTL and ATL* for discrete systems. We now turn our attention to *probabilistic computation tree logic* pCTL* [50; 51; 52] which describes behavior of probabilistic systems. The syntax of pCTL* is:

$$\begin{array}{lll} \text{state formulas:} & \psi & ::= ap \mid \psi \wedge \psi \mid \neg\psi \mid \mathbb{P}_I(\varphi) \\ \text{path formulas:} & \varphi & ::= \psi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi\mathcal{U}\varphi, \end{array}$$

where $ap \in Ap$ and $I \leq [0,1]$ is an interval with rational bounds.

The semantics of pCTL* path formulas with respect to a play of an MDP $\mathcal{M}$ is defined as follows:

$$\begin{array}{lll} \rho \models \psi & \text{iff} & \rho[0] \models \psi \\ \rho \models \varphi_1 \wedge \varphi_2 & \text{iff} & \rho \models \varphi_1 \text{ and } \rho \models \varphi_2 \\ \rho \models \bigcirc\varphi & \text{iff} & \rho_1 \models \varphi \\ \rho \models \varphi_1 \mathcal{U} \varphi_2 & \text{iff} & \exists j \in \mathbb{N} : (\rho_j \models \varphi_2 \text{ and } \forall 0 \leq i < j : \rho_i \models \varphi_1) \end{array}$$

Given a path formula φ and an MDP $\mathcal{M}$, we denote by $[\![\varphi]\!]_{\mathcal{M}}$ (or simply $[\![\varphi]\!]$ if the MDP is clear from the context) the set of plays ρ in $\mathcal{M}$ such that $\rho \models \varphi$. The semantics of state formulas is as follows:

$$\begin{array}{lll} s \models ap & \text{iff} & ap \in L(s) \\ s \models \psi_1 \wedge \psi_2 & \text{iff} & s \models \psi_1 \text{ and } s \models \psi_2 \\ s \models \neg\varphi & \text{iff} & \text{not } s \models \varphi \\ s \models \mathbb{P}_I(\varphi) & \text{iff} & \forall \sigma \in \Sigma \ : \ \mathbb{P}^{\sigma}_s([\![\varphi]\!]) \in I. \end{array}$$

2.3 Statistical Inference

Statistical inference allows us to reason about probabilistic systems, such as Markov chains, by repeatedly observing their behavior. Statistical problems can be phrased in a qualitative setting (as *hypothesis testing*) or in quantitative setting (as *confidence intervals*). In hypothesis testing we ask whether the probability of an event is below or above a user-specified threshold. In contrast, a confidence interval gives us a value range that captures the probability of the event with high confidence.

2.3.1 Hypothesis Testing

Let X be a random variable, and suppose we are interested whether the expected value $\mathbb{E}(X)$ is larger or smaller than some threshold p. We formulate this question as a hypothesis testing problem, where we decide between the *null hypothesis* H_0 and the *alternative hypothesis* H_1:

$$H_0 : \mathbb{E}(X) \geq p + \varepsilon \qquad\qquad H_1 : \mathbb{E}(X) < p - \varepsilon. \tag{2.1}$$

The indifference region $\varepsilon \geq 0$ describes the interval $[p - \varepsilon, p + \varepsilon)$ were both hypothesis are acceptable.

Test strength Two types of errors are used to evaluate precision of a solution. A *type I error* is the probability of accepting H_1 when H_0 holds. Similarly, a *type II error* is the probability of choosing H_0 when H_1 holds. The *test strength* (α, β) is a pair of values that bound the maximum probabilities of making type I and type II errors, respectively. In general, it is not possible to obtain low values of α and β at the same time when the indifference region ε is zero, since the probability $\mathbb{E}(X)$ may be arbitrary close to the threshold from either side, making type I or II error very likely.

Sequential probability ratio test The *sequential probability ratio test* (SPRT) is a popular statistical procedure for hypothesis testing [53; 54]. In the SPRT the number of samples is not fixed, but sampling continues until the observations give strong evidence in favor of H_0 or H_1. The SPRT gives no guarantee on the maximal number of samples; in practice, however, it often terminates quickly.

The SPRT works as follows. Suppose X is Bernoulli random variable, i.e. only values 0 and 1 are possible. After observing samples $\mathbf{x} = x_1, \ldots, x_n$ from X the following ratio is computed:

$$\frac{\mathbb{P}(\mathbf{x}|p_1)}{\mathbb{P}(\mathbf{x}|p_0)} = \prod_{i=1}^{n} \frac{\mathbb{P}(X = x_i \mid \mathbb{E}(X) = p_1)}{\mathbb{P}(X = x_i \mid \mathbb{E}(X) = p_0)} = \frac{p_1^{d_n}(1-p_1)^{n-d_n}}{p_0^{d_n}(1-p_0)^{n-d_n}},$$

where $d_n = \sum_{i=1}^{n} x_i$, $p_0 = p + \varepsilon$, and $p_1 = p - \varepsilon$. The decision rule for accepting a hypothesis is:

$$\text{accept } H_0 \text{ if } \frac{\mathbb{P}(\mathbf{x}|p_1)}{\mathbb{P}(\mathbf{x}|p_0)} \leq B \qquad \text{accept } H_1 \text{ if } \frac{\mathbb{P}(\mathbf{x}|p_1)}{\mathbb{P}(\mathbf{x}|p_0)} \geq A. \tag{2.2}$$

Finding the values of A, B such the test has the required strength is a difficult task. In practice, values $A = (1-\beta)/\alpha$ and $B = \beta/(1-\alpha)$ are used, since they result in a test whose strength is close to (α, β) [54].

2.3.2 Confidence Intervals

The goal of confidence interval estimation is to provide an interval that captures the expected value $\mathbb{E}(X)$ with high probability. Given observations $x_1, \ldots, x_n$ from X and a confidence level α, we wish to construct an interval $[a, b]$ that captures the expected value with probability at least $1 - \alpha$:

$$\mathbb{P}(a \leq \mathbb{E}(X) \leq b) \geq 1 - \alpha.$$

Often, we are given a bound on the difference $b - a$, and wish to generate enough samples to construct an interval of the given size and confidence.

Normal-based estimation We present a simple method for constructing a confidence interval that is based on the central-limit theorem [55]. The interval constructed by this method is not guaranteed to be correct, however the precision of the interval increases with the number of samples. In practice, this method gives good results for large number of samples.

Suppose we have observed values $x_1, \ldots, x_n$ from X. We define the following two estimators:

$$\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad S_n^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{X}_n)^2.$$

The $1 - \alpha$ confidence interval for $\mathbb{E}(X)$ is then

$$[\bar{X}_n - \Phi(\alpha/2)S_n, \bar{X}_n - \Phi(\alpha/2)S_n],$$

where Φ is the CDF of the standard normal distribution:

$$\Phi(x) \;=\; \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-t^2}{2}} dt.$$

3 Statistical Model Checking for Unbounded Temporal Properties

3.1 Introduction

Traditional numerical algorithms for the verification of Markov chains may be computationally intense or inapplicable, when facing a large state space or limited knowledge of the chain. To this end, statistical algorithms are used as a powerful alternative. *Statistical model checking* (SMC) typically refers to approaches where (i) finite paths of the Markov chain are sampled a finite number of times, (ii) the property of interest is verified for each sampled path (e.g. state r is reached), and (iii) hypothesis testing or statistical estimation is used to infer conclusions (e.g. state r is reached with probability at most 0.5) and give statistical guarantees (e.g. the conclusion is valid with 99% confidence). SMC approaches differ in (a) the class of properties they can verify (e.g. bounded or unbounded properties), (b) the strength of statistical guarantees they provide (e.g. confidence bounds, only asymptotic convergence of the method towards the correct value, or none), and (c) the amount of information they require about the Markov chain (e.g. the topology of the graph). In this chapter, we provide an algorithm for SMC of unbounded properties, with confidence bounds, in the setting where only the minimum transition probability of the chain is known. Such an algorithm is particularly desirable in scenarios when the system is not known ("black box"), but also when it is too large to construct or fit into memory.

Most of the previous efforts in SMC have focused on the analysis of properties with bounded horizon [56; 57; 58; 59; 17; 60; 61]. For bounded properties (e.g. state r is reached with probability at most 0.5 in the first 1000 steps) statistical guarantees can be obtained in a completely black-box setting, where execution runs of the Markov chain can be observed, but no other

LTL, mean payoff	×	**here**	[31](LTL)		
$\Diamond$, **U**	×	**here**	—”—	[30]	e.g. [34]
bounded	e.g. [56]				
	no info	p_{min}	$\lvert S\rvert, p_{\mathsf{min}}$	λ	topology

Table 3.1: SMC approaches to Markov chain verification, organised by (i) the class of verifiable properties, and (ii) by the required information about the Markov chain, where p_{min} is the minimum transition probability, $|S|$ is the number of states, and λ is the second largest eigenvalue of the chain.

information about the chain is available. Unbounded properties (e.g. state r is reached with probability at most 0.5 in any number of steps) are significantly more difficult, as a stopping criterion is needed when generating a potentially infinite execution run, and some information about the Markov chain is necessary for providing statistical guarantees (for an overview, see Table 3.1). On the one hand, some approaches require the knowledge of the full topology in order to preprocess the Markov chain. On the other hand, when the topology is not accessible, there are approaches where the correctness of the statistics relies on information ranging from the second eigenvalue λ of the Markov chain, to the knowledge of both the number $|S|$ of states and the minimum transition probability p_{min}.

Our contribution is a new SMC algorithm for full linear temporal logic (LTL), as well as for unbounded quantitative properties (mean payoff), which provides strong statistical guarantees (see Chapter 2 for LTL and mean payoff). Our algorithm uses less information about the Markov chain than previous algorithms that provide confidence bounds for unbounded properties—we need to know only the minimum transition probability p_{min} of the chain, and not the number of states nor the topology. Yet, experimentally, our algorithm performs in many cases better than these previous approaches (see Section 5). Our main idea is to *monitor each execution run on the fly in order to build statistical hypotheses about the structure of the Markov chain*. In particular, if from observing the current prefix of an execution run we can stipulate that with high probability a bottom strongly connected component (BSCC) of the chain has been entered, then we can terminate the current execution run (see Section 2.2.2 for the definition of BSCCs). The information obtained from execution prefixes allows us to terminate executions as soon as the property is decided with the required confidence, which is usually much earlier than any

bounds that can be computed a priori. As far as we know, this is the first SMC algorithm that uses information obtained from execution prefixes.

Finding p_{min} is a light assumption in many realistic scenarios and often does not depend on the size of the chain – e.g. bounds on the rates for reaction kinetics in chemical reaction systems are typically known; alternatively, from a PRISM [62] language model they can be easily inferred without constructing the respective state space.

We illustrate our key idea on the following example.

Example 3.1. *Consider the property of reaching state r in the Markov chain depicted in Figure 3.1. While the execution runs reaching r satisfy the property and can be stopped without ever entering any v_i, the finite execution paths without r, such as $stuttutuut$, are inconclusive. In other words, observing this path does not rule out the existence of a transition from, e.g., u to r, which, if existing, would eventually be taken with probability 1. This transition could have arbitrarily low probability, rendering its detection arbitrarily unlikely, yet its presence would change the probability of satisfying the property from 0.5 to 1. However, knowing that if there exists such a transition leaving the set, its transition probability is at least $p_{\mathsf{min}} = 0.01$, we can estimate the probability that the system is stuck in the set $\{t, u\}$ of states. Indeed, if existing, the exit transition was missed at least four times, no matter whether it exits t or u. Consequently, the probability that there is no such transition and $\{t, u\}$ is a BSCC is at least $1 - (1 - p_{\mathsf{min}})^4$.*

This means that, in order to get 99% confidence that $\{t, u\}$ is a BSCC, we only need to see both t and u around 500 times[1] *on a run. This is in stark contrast to a priori bounds that provide the same level of confidence, such as the $(1/p_{\mathsf{min}})^{|S|} = 100^{\mathcal{O}(m)}$ runs required by [31], which is infeasible for large m. In contrast, our method's performance is independent of m.*

Monitoring execution prefixes allows us to design an SMC algorithm for complex unbounded properties such as full LTL. More precisely, we present a new SMC algorithm for LTL over Markov chains, specified as the following hypothesis testing problem (see Section 2.3.1):

Input:

- we can sample finite runs of arbitrary length from an unknown finite-state discrete-time Markov chain $\mathcal{M}$ according to the initial distribution[2],

[1] $1 - (1 - p_{\mathsf{min}})^{500} = 1 - 0.99^{500} \approx 0.993$

[2] We have a black-box system in the sense of [57], different from e.g. [56] or [63], where simulations can be run from any state.

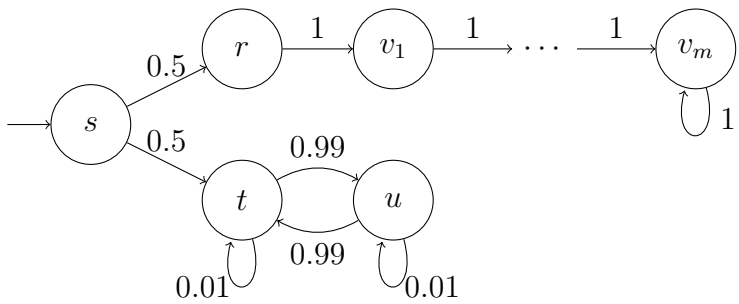

Figure 3.1: A Markov chain.

- we are given a lower bound $p_{\mathsf{min}} > 0$ on the transition probabilities in $\mathcal{M}$,
- an LTL formula φ,
- a threshold probability p,
- an indifference region $\varepsilon > 0$,
- two error bounds $\alpha, \beta > 0$.

Output:

- if $\mathbb{P}[\varphi] \geq p + \varepsilon$, return YES with probability at least $1 - \alpha$, and
- if $\mathbb{P}[\varphi] \leq p - \varepsilon$, return NO with probability at least $1 - \beta$.

In addition, we present the first SMC algorithm for computing the mean payoff of Markov chains whose states are labelled with rewards.

Our idea of inferring the structure of the Markov chain on the fly, while generating execution runs, allows for their early termination. In Section 3.6 we will see that for many chains arising in practice, such as the concurrent probabilistic protocols from the PRISM benchmark suite [64], the BSCCs are reached quickly and, even more importantly, can be small even for very large systems. Consequently, many execution runs can be stopped quickly. Moreover, the number of execution runs necessary for a required precision and confidence is independent of the size of the state space, therefore this number can be small even for highly confident results (a good analogy is that of the opinion polls: the precision and confidence of opinion polls is regulated by the sample size and is independent of the size of the population). It is therefore not surprising that, experimentally, in most cases from the benchmark suite, our method outperforms previous

methods (often even the numerical methods) despite requiring much less knowledge of the Markov chain, and despite providing strong guarantees in the form of confidence bounds. In Section 3.7, we also provide theoretical bounds on the running time of our algorithm for classes of Markov chains on which it performs particularly well.

Outline The chapter is organised as follows. In Section 3.2 we review the related work. In Section 3.3 we describe our SMC method for unbounded reachability. Sections 3.4 and 3.5 present extensions to linear temporal logic and mean payoff, respectively. Section 3.6 describes experimental evaluation of our method. Finally, in Section 3.7 we give a theoretical bound on the expected running time of our algorithms.

3.2 Related Work

Most of the effort in statistical model checking methods has focused on the analysis of properties with bounded horizon, e.g., [56; 57; 58; 59; 60; 61]. These are properties whose satisfaction on a run can be decided based on its prefix of a fixed length. The unbounded properties are often investigated under the name "unbounded until" [34; 30], which is only a slight generalisation of reachability.

SMC of unbounded properties was first considered in [65]. It was suggested to try longer and longer simulations, but no bounds when to stop this process were given. The first solution was proposed in [33]. A simulation is to be stopped whenever we reach a point from which the goal state r cannot be reached at all. To this end, another set of simulations is run from such potential point to determine if there is any path to r. In order to avoid infinite simulations here, the simulations are stopped in each step with some "termination probability" p_{term}. This transforms the hypothesis testing task to one where simulations are almost surely finite. It was observed in [59] that this transformation works only on Markov chains that do not contain loops.

In [32] the probability of unbounded property is approximated by a bounded variant that is sufficiently long. The correctness of this approach requires the second eigenvalue to be computed, which is as hard as the verification problem itself. A completely different approach is taken in [63]. Using coupling methods one can estimate the stationary distribution. However, the method is limited to ergodic Markov chains. In such a case all states of the system will be reached almost

surely (and infinitely often).

Notably, in [30] two approaches are described. The first approach proposes to terminate sampled paths at every step with some probability p_{term}. In order to guarantee the asymptotic convergence of this method the second eigenvalue λ of the chain must be known, similar to [32]. It should be noted that their method provides only asymptotic guarantees as the width of the confidence interval converges to zero. The second approach of [30] requires the knowledge of the chain's topology, which is used to transform the chain so that all potentially infinite paths are eliminated.

In [34] another transformation is performed, again requiring knowledge of the topology. This transformation assigns equal probability to all transitions leaving from a state, which effectively reduces checking of an unbounded until to a bounded variant. This method can only be used to check whether a property holds with a positive probability, but does not allow one to estimate the probability.

The (pre)processing of the state space required by the topology-aware methods, as well as by traditional numerical methods for Markov chain analysis, is a major practical hurdle for large (or unknown) state spaces. In [31] a priori bounds for the length of execution runs are calculated from the minimum transition probability and the number of states. However, without taking execution information into account, these bounds are exponential in the number of states and highly impractical, as illustrated in the example above.

There are also extensions of SMC to timed systems [66]. Our approach is also related to [67; 68], where the product of a non-deterministic system and Büchi automaton is explored for accepting lassos. We are not aware of any method for detecting BSCCs by observing a single run, employing no directed search of the state space.

To the best of our knowledge, we present the first SMC algorithm that provides confidence bounds for unbounded qualitative properties with access to only the minimum probability of the chain p_{min}, and the first SMC algorithm for quantitative properties.

3.3 Solution for Reachability

A fundamental problem in Markov chain verification is computing the probability that a certain set of goal states is reached. For the rest of the chapter, let $\mathcal{M} = (S, \mathbf{P}, L, \mu)$ be a Markov chain

and $G \subseteq S$ be the set of the goal states in $\mathcal{M}$. We write

$$p_{\mathsf{min}} = \min(\{\mathbf{P}(s, s') > 0 \mid s, s' \in S\})$$

to denote the smallest positive transition probability in $\mathcal{M}$. We denote the event "eventually a state in G is reached" in an LTL-like notation as

$$\Diamond G = \{\rho \in \mathsf{Runs} \mid \exists i \geq 0 : L(\rho[i]) \in G\}.$$

Our goal is to estimate the probability $\mathbb{P}(\Diamond G)$ by a statistical algorithm. Since no bound on the number of steps for reaching G is given, the major difficulty for any statistical approach is to decide how long each sampled path should be. We can stop extending the path either when we reach G, or when no more new states can be reached anyways. The latter happens if and only if we are in a BSCC and we have seen all of its states.

In this section, we first show how to monitor each simulation run on the fly, in order to detect quickly if a BSCC has been entered with high probability. Then, we show how to use hypothesis testing in order to estimate $\mathbb{P}(\Diamond G)$.

3.3.1 BSCC Detection

Error bound on a single candidate We start with an example illustrating how to measure the probability of reaching a BSCC from one path observation.

Example 3.2. *Recall Example 3.1 and Figure 3.1. Now, consider an execution path* $stuttutu$*. Intuitively, does* $\{t, u\}$ *look like a good "candidate" for being a BSCC of* $\mathcal{M}$*? We visited both* t *and* u *three times; we have taken a transition from each* t *and* u *at least twice without leaving* $\{t, u\}$*. By the same reasoning as in Example 3.1, we could have missed some outgoing transition with probability at most* $(1 - p_{\mathsf{min}})^2$*. The structure of the system that can be deduced from this path is in Figure 3.2 and is correct with probability at least* $1 - (1 - p_{\mathsf{min}})^2$*.*

Now we formalise our intuition. Given a finite or infinite sequence $\rho = s_0 s_1 \cdots$, the *support* of ρ is the set $\mathrm{Supp}(\rho) = \{s_0, s_1, \ldots\}$. Further, the *graph of* ρ is given by vertices $\mathrm{Supp}(\rho)$ and edges $\{(s_i, s_{i+1}) \mid i = 0, 1, \ldots\}$.

Definition 3.1 (Candidate)**.** *If a path* π *has a suffix* κ *such that* $\mathrm{Supp}(\kappa)$ *is a BSCC of the graph of* π*, we call* $\mathrm{Supp}(\kappa)$ *the* candidate of π*. Moreover, for* $k \in \mathbb{N}$*, we call it a* k-candidate (of π) *if*

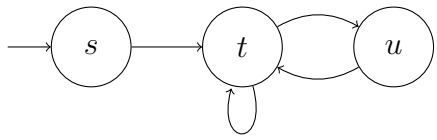

Figure 3.2: The graph of a path $stuttutu$.

each $s \in \mathrm{Supp}(\kappa)$ *has at least* k *occurrences in* κ *and the last element of* κ *has at least* $k+1$ *occurrences. A* k-candidate of a run ρ *is a* k*-candidate of some prefix of* ρ*.*

Note that for each path there is at most one candidate. Therefore, we write $K(\pi)$ to denote the candidate of π if there is one, and $K(\pi) = \perp$, otherwise. Observe that each $K(\pi) \neq \perp$ is strongly connected in $\mathcal{M}$.

Example 3.3. *Consider a path* $\pi = stuttutu$*, then* $K(\pi) = \{t, u\}$*. Observe that* $\{t\}$ *is not a candidate as it is not maximal. Further,* $K(\pi)$ *is a* 2*-candidate (and as such also a* 1*-candidate), but not a* 3*-candidate. Intuitively, the reason is that we only took a transition from* u *(to the candidate) twice, cf. Example 3.2.*

Intuitively, the higher the k the more it looks as if the k-candidate is indeed a BSCC. Denoting by $Cand_k(K)$ the random predicate of K being a k-candidate on a run, the probability of "unluckily" detecting any specific non-BSCC set of states K as a k-candidate, can be bounded as follows.

Lemma 3.1. *For every* $K \subseteq S$ *such that* $K \notin \mathsf{BSCC}$*, and every* $s \in K$*,* $k \in \mathbb{N}$*,*

$$\mathbb{P}_s[Cand_k(K)] \leq (1 - p_{\mathsf{min}})^k.$$

Proof. Since K is not a BSCC, there is a state $t \in K$ with a transition to $t' \notin K$. The set of states K becomes a k-candidate of a run starting from s, only if t is visited at least k times by the path and was never followed by t' (indeed, even if t is the last state in the path, by definition of a k-candidate, there are also at least k previous occurrences of t in the path). Further, since the transition from t to t' has probability at least p_{min}, the probability of not taking the transition k times is at most $(1 - p_{\mathsf{min}})^k$. □

Example 3.4. *We illustrate how candidates "evolve over time" along a run. Consider a run* $\rho = s_0 s_0 s_1 s_0 \cdots$ *of the Markov chain in Figure 3.3. The empty and one-letter prefix do*

not have the candidate defined, s_0s_0 *has a candidate* $\{s_0\}$*, then again* $K(s_0s_0s_1) = \bot$*, and* $K(s_0s_0s_1s_0) = \{s_0, s_1\}$*. One can observe that subsequent candidates are either disjoint or contain some of the previous candidates. Consequently, there are at most* $2|S| - 1$ *candidates on every run, which is in our setting an unknown bound.*

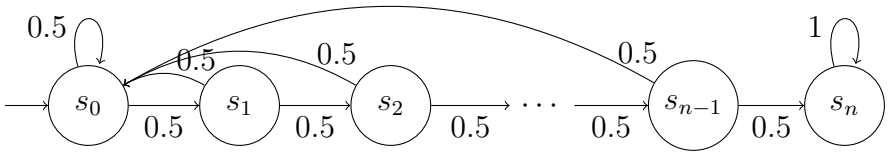

Figure 3.3: A family (for $n \in \mathbb{N}$) of Markov chains with large eigenvalues.

Error bound on multiple candidates While we have bounded the probability of detecting any specific non-BSCC set K as a k-candidate, we need to bound the overall error for detecting a candidate that is not a BSCC. Since there can be many false candidates on a run before the real BSCC (e.g. Figure 3.3), we need to bound the error of reporting any of them.

In the following, we first formalise the process of discovering candidates along the run. Second, we bound the error that any of the non-BSCC candidates becomes a k-candidate. Third, we bound the overall error of not detecting the real BSCC by increasing k every time a different candidate is found.

We start with discovering the sequence of candidates on a run. For a run $\rho = s_0s_1\cdots$, consider the sequence of random variables defined by $K(s_0 \dots s_j)$ for $j \geq 0$, and let $(K_i)_{i\geq 1}$ be the subsequence without undefined elements and with no repetition of consecutive elements. For example, for a run $\rho = s_0s_1s_1s_1s_0s_1s_2s_2\cdots$, we have $K_1 = \{s_1\}$, $K_2 = \{s_0, s_1\}$, $K_3 = \{s_2\}$, etc. Let K_j be the last element of this sequence, called the *final candidate*. Additionally, we define $K_\ell = K_j$ for all $\ell > j$. We describe the lifetime of a candidate. Given a non-final K_i, we write $\rho = \alpha_i\beta_ib_i\gamma_id_i\delta_i$ so that $\mathrm{Supp}(\alpha_i) \cap K_i = \emptyset$, $\mathrm{Supp}(\beta_ib_i\gamma_i) = K_i$, $d_i \notin K_i$, and $K(\alpha_i\beta_i) \neq K_i$, $K(\alpha_i\beta_ib_i) = K_i$. Intuitively, we start exploring K_i in β_i; K_i becomes a candidate in b_i, the birthday of the ith candidate; it remains to be a candidate until d_i, the death of the ith candidate. For example, for the run $\rho = s_0s_1s_1s_1s_0s_1s_2s_2\cdots$ and $i = 1$, $\alpha_1 = s_0$, $\beta_1 = s_1$, $b_1 = s_1$, $\gamma_1 = s_1$, $d_1 = s_0$, $\delta_1 = s_1s_2s_2\rho[8]\rho[9]\cdots$. Note that the final candidate is almost surely a BSCC of $\mathcal{M}$ and would thus have γ_j infinite.

Now, we proceed to bounding errors for each candidate. Since there is an unknown number of candidates on a run, we will need a slightly stronger definition. First, observe that $Cand_k(K_i)$ iff K_i is a k-candidate of $\beta_i b_i \gamma_i$. We say K_i is a *strong k-candidate*, written $SCand_k(K_i)$, if it is a k-candidate of $b_i\gamma_i$. Intuitively, it becomes a k-candidate even not counting the discovery phase. As a result, even if we already assume there exists an ith candidate, its strong k-candidacy gives the guarantees on being a BSCC as above in Lemma 3.1.

Lemma 3.2. *For every $i, k \in \mathbb{N}$, we have*

$$\mathbb{P}(SCand_k(K_i) \mid K_i \notin \mathsf{BSCC}) \leq (1 - p_{\mathsf{min}})^k .$$

Proof.

$$\begin{aligned}
&\mathbb{P}(SCand_k(K_i) \mid K_i \notin \mathsf{BSCC}) \\
&= \frac{\mathbb{P}(SCand_k(K_i), K_i \notin \mathsf{BSCC})}{\mathbb{P}(K_i \notin \mathsf{BSCC})} \\
&= \frac{1}{\mathbb{P}(K_i \notin \mathsf{BSCC})} \sum_{\substack{C \in \mathsf{SC} \setminus \mathsf{BSCC} \\ s \in C}} \mathbb{P}(SCand_k(C), K_i = C, b_i = s) \\
&= \frac{1}{\mathbb{P}(K_i \notin \mathsf{BSCC})} \sum_{\substack{C \in \mathsf{SC} \setminus \mathsf{BSCC} \\ s \in C}} \mathbb{P}(K_i = C, b_i = s)\mathbb{P}_s(Cand_k(C)) && \text{(by Markov property)} \\
&\leq \frac{1}{\mathbb{P}(K_i \notin \mathsf{BSCC})} \sum_{\substack{C \in \mathsf{SC} \setminus \mathsf{BSCC} \\ s \in C}} \mathbb{P}(K_i = C, b_i = s)(1 - p_{\mathsf{min}})^k && \text{(by Lemma 3.1)} \\
&= (1 - p_{\mathsf{min}})^k . && \text{(since } \mathbb{P}(K_i \notin \mathsf{BSCC}) = \textstyle\sum_{\substack{C \in \mathsf{SC} \setminus \mathsf{BSCC} \\ s \in C}} \mathbb{P}(K_i = C, b_i = s))
\end{aligned}$$

□

Since the number of candidates can only be bounded with some knowledge of the state space, e.g. its size, we assume no bounds and provide a method to bound the error even for an unbounded number of candidates on a run.

Lemma 3.3. *For $(k_i)_{i=1}^{\infty} \in \mathbb{N}^{\mathbb{N}}$, let $\mathcal{E}rr$ be the set of runs such that for some $i \in \mathbb{N}$, we have $SCand_{k_i}(K_i)$ despite $K_i \notin \mathsf{BSCC}$. Then*

$$\mathbb{P}(\mathcal{E}rr) < \sum_{i=1}^{\infty} (1 - p_{\mathsf{min}})^{k_i}.$$

Proof.

$$
\begin{aligned}
\mathbb{P}(\mathcal{E}rr) &= \mathbb{P}\left[\bigcup_{i=1}^{\infty}\Big(SCand_{k_i}(K_i) \cap K_i \notin \mathsf{BSCC}\Big)\right) \\
&\leq \sum_{i=1}^{\infty} \mathbb{P}(SCand_{k_i}(K_i) \cap K_i \notin \mathsf{BSCC}) && \text{(by the union bound)} \\
&= \sum_{i=1}^{\infty} \mathbb{P}(SCand_{k_i}(K_i) \mid K_i \notin \mathsf{BSCC}) \cdot \mathbb{P}(K_i \notin \mathsf{BSCC}) \\
&\leq \sum_{i=1}^{\infty} \mathbb{P}(SCand_{k_i}(K_i) \mid K_i \notin \mathsf{BSCC}) \\
&= \sum_{i=1}^{\infty} (1 - p_{\mathsf{min}})^{k_i}. && \text{(by Lemma 3.2)}
\end{aligned}
$$

□

Algorithm for BSCC detection In Algorithm 3.1 we present a procedure for deciding whether a BSCC inferred from a path π is indeed a BSCC with confidence greater than $1 - \delta$. We use notation $\textsc{SCand}_{k_i}(K, \pi)$ to denote the function deciding whether K is a strong k_i-candidate on π. The overall error bound is obtained by setting $k_i = \frac{i - \log \delta}{-\log(1 - p_{\mathsf{min}})}$.

Algorithm 3.1 REACHEDBSCC

Input: path $\pi = s_0 s_1 \cdots s_n$, $p_{\mathsf{min}}, \delta \in (0, 1]$

Output: Yes iff $K(\pi) \in \mathsf{BSCC}$

```
C ← ⊥, i ← 0
for j = 0 to n do
    if K(s_0 ··· s_j) ≠ ⊥ and K(s_0 ··· s_j) ≠ C then
        C ← K(s_0 ··· s_j)
        i ← i + 1
k_i ← (i − log δ) / (− log(1 − p_min))
if i ≥ 1 and SCand_{k_i}(K(π), π) then return Yes
else return No
```

Theorem 3.1. *For every $\delta > 0$, Algorithm 3.1 is correct with error probability at most δ.*

Proof. Since M is finite, the Algorithm 3.1 terminates almost surely. The probability to return an incorrect result can be bounded by returning incorrect result for one of the non-final candidates, which by Lemma 3.3 is as follows:

$$\sum_{i=1}^{\infty}(1-p_{\mathsf{min}})^{k_i} = \sum_{i=1}^{\infty}(1-p_{\mathsf{min}})^{\frac{-i+\log\delta}{\log(1-p_{\mathsf{min}})}} = \sum_{i=1}^{\infty}2^{-i+\log\delta} = \sum_{i=1}^{\infty}\delta/2^i = \delta.$$

□

Sampling algorithm We have shown how to detect a BSCC of a single path with desired confidence. In Algorithm 3.2, we show how to use our BSCC detection method to decide whether a given path reaches the set G with confidence $1-\delta$. The function $\mathsf{NextState}(\pi)$ randomly picks a state according to the initial distribution μ if the path is empty ($\pi = \lambda$); otherwise, if ℓ is the last state of π, it randomly chooses its successor according to $\mathbf{P}(\ell, \cdot)$. The algorithm returns **Yes** when π reaches a state in G, and **No** when for some i, the ith candidate is a strong k_i-candidate. In the latter case, with probability at least $1-\delta$, π has reached a BSCC not containing G. Hence, with probability at most δ, the algorithm returns **No** for a path that could reach a goal.

Algorithm 3.2 SINGLEPATHREACH

Input: goal states G of $\mathcal{M}$, $p_{\mathsf{min}}, \delta \in (0, 1]$

Output: **Yes** iff a run reaches G

```
π ← λ
repeat
    s ← NextState(π)
    π ← π · s
    if s ∈ G then return Yes                ▷ We have provably reached G
until ReachedBSCC(π, p_min, δ)
return No                                   ▷ By Theorem 3.1, ℙ(K(π) ∈ BSCC) ≥ 1 − δ
```

3.3.2 Hypothesis Testing With Bounded Error

In the following, we show how to estimate the probability of reaching a set of goal states, by combining the BSCC detection and hypothesis testing. More specifically, we sample many paths of a Markov chain, decide for each whether it reaches the goal states (Algorithm 3.2), and then

use hypothesis testing to estimate the event probability. The hypothesis testing is adapted to the fact that testing reachability on a single path may report false negatives.

Let $X^\delta_\Diamond$ be a Bernoulli random variable, such that $X^\delta_\Diamond = 1$ if and only if

$$\textsc{SinglePathReach}(G, p_{\mathsf{min}}, \delta) = \textbf{Yes},$$

describing the outcome of Algorithm 3.2. The following theorem establishes that $X^\delta_\Diamond$ estimates $\mathbb{P}(\Diamond G)$ with a bias bounded by δ.

Theorem 3.2. *For every $\delta > 0$, we have $\mathbb{P}(\Diamond G) - \delta \leq \mathbb{E}(X^\delta_\Diamond) \leq \mathbb{P}(\Diamond G)$.*

Proof. Since the event $\Diamond G$ is necessary for $X^\delta_\Diamond = 1$, we have $\mathbb{P}(\Diamond G \mid X^\delta_\Diamond = 1) = 1$. It follows that $\mathbb{E}(X^\delta_\Diamond) = \mathbb{P}(X^\delta_\Diamond = 1) = \mathbb{P}(\Diamond G, X^\delta_\Diamond = 1) \leq \mathbb{P}(\Diamond G)$, hence the upper bound. As for the lower bound:

$$\begin{aligned}
\mathbb{E}(X^\delta_\Diamond) = \mathbb{P}(X^\delta_\Diamond = 1) &= \mathbb{P}(\Diamond G, X^\delta_\Diamond = 1) && \Diamond G \text{ is necessary for } X^\delta_\Diamond = 1 \\
&= \mathbb{P}(\Diamond G) - \mathbb{P}(\Diamond G, X^\delta_\Diamond = 0) \\
&\geq \mathbb{P}(\Diamond G) - \delta. && \text{by Theorem 3.1}
\end{aligned}$$

□

In order to conclude on the value $\mathbb{P}(\Diamond G)$, the standard statistical model checking approach via hypothesis testing (see Section 2.3.1) decides between the hypothesis

$$H_0 : \mathbb{P}(\Diamond G) \geq p + \varepsilon \qquad\qquad H_1 : \mathbb{P}(\Diamond G) < p - \varepsilon.$$

where ε is a desired indifference region. As we do not have precise observations on each path, we reduce this problem to a hypothesis testing on the variable $X^\delta_\Diamond$ with a narrower indifference region:

$$H'_0 : \mathbb{E}(X^\delta_\Diamond) \geq p + (\varepsilon - \delta) \qquad\qquad H'_1 : \mathbb{E}(X^\delta_\Diamond) < p - \varepsilon,$$

for some $\delta < \varepsilon$.

We define the reduction simply as follows. Given a statistical test $\mathcal{T}'$ for H'_0, H'_1 we define a test $\mathcal{T}$ that accepts H_0 if $\mathcal{T}'$ accepts H'_0, and H_1 otherwise. The following lemma shows that $\mathcal{T}$ has the same strength as $\mathcal{T}'$.

Lemma 3.4. *Suppose the test $\mathcal{T}'$ decides between H'_0 and H'_1 with strength (α, β). Then the test $\mathcal{T}$ decides between H_0 with H_1 with strength (α, β).*

Proof. Consider type I error of $\mathcal{T}$. Assume that H_0 holds, which means $\mathbb{P}(\Diamond G) \geq p + \varepsilon$. By Theorem 3.2 it follows that $\mathbb{P}(X^\delta_\Diamond = 1) \geq \mathbb{P}(\Diamond G) - \delta \geq p + (\varepsilon - \delta)$, thus H'_0 also holds. By assumption the test $\mathcal{T}'$ accepts H'_1 with probability at most α, thus, by the reduction, $\mathcal{T}$ also accepts H_1 with probability $\leq \alpha$. The proof for type II error is analogous. □

Lemma 3.4 gives us the following algorithm to decide between H_0 and H_1. We generate samples $x_0, x_1, \cdots, x_n \sim X^\delta_\Diamond$ from $\textsc{SinglePathReach}(G, p_{\mathsf{min}}, \delta)$, and apply a statistical test to decide between H'_0 and H'_1. Finally, we accept H_0 if H'_0 was accepted by the test, and H_1 otherwise.

3.4 Solution for Linear Temporal Logic

We show how our method extends to properties expressible by LTL (see Section 2.1.2), in the same manner, to all ω-regular properties. Given a Markov chain $\mathcal{M} = (S, \mathbf{P}, L, \mu)$ and an LTL formula φ, we are interested in the measure

$$\mathbb{P}_{\mathcal{M}}(\varphi) \;=\; \mathbb{P}_{\mathcal{M}}(\{\rho \in \mathsf{Runs} \mid L(\rho) \models \varphi\}),$$

where L is naturally extended to runs by $L(\rho)[i] = L(\rho[i])$ for all i.

The probability of $\mathbb{P}_{\mathcal{M}}(\varphi)$ can be reduced to reachability probability in the product of $\mathcal{M}$ and a deterministic Rabin automaton (cf. Section 2.1.3). Recall that for every LTL formula φ, a DRA $\mathcal{A}_\varphi$ can be effectively constructed that accepts the same same language as φ. The product of a MC and DRA is defined in the following way.

Definition 3.2 (Product of a MC and DRA)**.** *The product of a Markov chain $\mathcal{M} = (S, \mathbf{P}, L, \mu)$ and deterministic Rabin automaton $\mathcal{A} = (Q, \gamma, q_o, Acc)$ is the Markov chain $\mathcal{M} \otimes \mathcal{A} = (S \times Q, \mathbf{P}', L, \mu')$, where*

- $\mathbf{P}'((s,q),(s',q')) = \begin{cases} \mathbf{P}(s,s') & \text{if } q' = \gamma(q, L(s')) \\ 0 & \text{otherwise,} \end{cases}$
- $L'((s,q)) = L(s)$,
- $\mu'((s,q)) = \begin{cases} \mu(s) & \text{if } q = \gamma(q_o, L(s)) \\ 0 & \text{othewise.} \end{cases}$

Note that $\mathcal{M} \otimes \mathcal{A}$ has the same smallest transition probability p_{min} as $\mathcal{M}$.

The crux of LTL probabilistic model checking relies on the fact that the probability of satisfying an LTL property φ in a Markov chain $\mathcal{M}$ equals the probability of reaching an accepting BSCC in the Markov chain $\mathcal{M} \otimes \mathcal{A}_\varphi$. Formally, a BSCC C of $\mathcal{M} \otimes \mathcal{A}_\varphi$ is *accepting* if for some $(E, F) \in Acc$ we have $C \cap (S \times E) = \emptyset$ and $C \cap (S \times F) \neq \emptyset$. Let $\mathsf{AccBSCC}$ denote the union of all accepting BSCCs in $\mathcal{M} \otimes \mathcal{A}_\varphi$. Then we obtain the following fact [10]:

Lemma 3.5. *For every Markov chain $\mathcal{M}$ and* LTL *formula φ, we have*

$$\mathbb{P}_{\mathcal{M}}(\varphi) = \mathbb{P}_{\mathcal{M} \otimes \mathcal{A}_\varphi}(\Diamond \mathsf{AccBSCC}).$$

Sampling algorithm Algorithm 3.3 simulates a path of the Markov chain and augments it with states of the DRA, to obtain a path in $\mathcal{M} \otimes \mathcal{A}_\varphi$. The process continues until the augmented path reaches a BSCC of the product. Finally, the algorithm checks whether the BSCC is accepting. Since the input used is a Rabin automaton, the method applies to all ω-regular properties.

Algorithm 3.3 SINGLEPATHLTL

Input: DRA $\mathcal{A} = (Q, 2^{Ap}, \gamma, q_o, Acc)$, $p_{\mathsf{min}}, \delta \in (0, 1]$

Output: **Yes** iff the final candidate is an accepting BSCC

$q \leftarrow q_o$, $\pi \leftarrow \lambda$

repeat

  $s \leftarrow \mathsf{NextState}(\pi)$

  $q \leftarrow \gamma(q, L(s))$

  $\pi \leftarrow \pi \cdot (s, q)$

until REACHEDBSCC$(\pi, p_{\mathsf{min}}, \delta)$ $\quad \triangleright\ \mathbb{P}(K(\pi) \in \mathsf{BSCC}) \geq 1 - \delta$

return $\exists (E, F) \in Acc : K(\pi) \cap (S \times E) = \emptyset \wedge K(\pi) \cap (S \times F) \neq \emptyset$

3.4.1 Hypothesis testing with bounded error

Let X_φ^δ be a Bernoulli random variable, such that $X_\varphi^\delta = 1$ if and only if

$$\text{SINGLEPATHLTL}(\mathcal{A}_\varphi, p_{\mathsf{min}}, \delta) = \textbf{Yes}$$

Since the BSCC must be reached and fully explored to classify it correctly, the error of the algorithm can now be both-sided.

Theorem 3.3. *For every* $\delta > 0$, $\mathbb{P}(\varphi) - \delta \leq \mathbb{E}(X_\varphi^\delta) \leq \mathbb{P}(\varphi) + \delta$.

Further, like in Section 3.3.2, we can reduce the hypothesis testing problem for

$$H_0 : \mathbb{P}(\varphi) \geq p + \varepsilon \qquad \text{and} \qquad H_1 : \mathbb{P}(\varphi) \leq p - \varepsilon$$

for any $\delta < \varepsilon$ to the following hypothesis testing problem on the observable X_φ^δ

$$H_0' : \mathbb{E}(X_\varphi^\delta) \geq p + (\varepsilon - \delta) \qquad \text{and} \qquad H_1' : \mathbb{E}(X_\varphi^\delta) \leq p - (\varepsilon - \delta)\,.$$

3.5 Solution for Mean Payoff

We show that our method extends also to quantitative properties, such as mean payoff. For the definition of Markov reward model and mean payoff see Section 2.2.3.

Let $\mathcal{M}$ be a Markov reward model with reward function $r : S \to [0, 1]$ and $\mathsf{MP}(s)$ be the mean payoff for the state s. All states in the same BSCC have equal mean payoff, and this value can be computed from the reward function r and transition probabilities in the BSCC. In general, for every state s the mean-payoff can be computed as

$$\mathsf{MP}(s) \;=\; \sum_{C \in \mathsf{BSCC}} \mathbb{P}_s(\Diamond C) \cdot \mathsf{MP}_C,$$

where MP_C is the mean payoff of runs ending in C. We have already shown how our method estimates $\mathbb{P}(\Diamond C)$. Now we show how it extends to estimating transition probabilities in BSCCs and thus the mean payoff.

Single path analysis First, we focus on a single path π that has reached a BSCC $C = K(\pi)$ and show how to estimate the transition probabilities $\mathbf{P}(s, s')$ for each $s, s' \in C$. Let $X_{s,s'}$ be the random variable denoting the event that $\mathsf{NextState}(s) = s'$. $X_{s,s'}$ is a Bernoulli variable with parameter $\mathbf{P}(s, s')$, so we use the obvious estimator $\hat{\mathbf{P}}(s, s') = \#_{ss'}(\pi)/\#_s(\pi)$, where $\#_\alpha(\pi)$ is the number of occurrences of α in π. If π is long enough so that $\#_s(\pi)$ is large enough, the estimation is guaranteed to have desired precision ξ with desired confidence $(1 - \delta_{s,s'})$. Indeed, using Höffding's inequality [55], we obtain

$$\mathbb{P}(\hat{\mathbf{P}}(s, s') - \mathbf{P}(s, s')| > \xi) \leq \delta_{s,s'} = 2e^{-2\#_s(\pi)\cdot\xi^2}\,. \tag{3.1}$$

Hence, we can extend the path π with candidate C until it is long enough so that we have a $1 - \delta_C$ confidence that all the transition probabilities in C are in the ξ-neighbourhood of our estimates, by ensuring that $\sum_{s,s' \in C} \delta_{s,s'} < \delta_C$. These estimated transition probabilities $\hat{\mathbf{P}}$ induce an *estimated mean payoff* $\hat{\mathsf{MP}}_C$. The following theorem relates the estimated and exact mean payoff.

Theorem 3.4. *Let C be a BSCC in a Markov chain $\mathcal{M}$ with rewards in the range $[0, 1]$, MP_C be the mean payoff of C, and $\hat{\mathsf{MP}}_C$ be the estimated mean payoff of C. Then*

$$|\hat{\mathsf{MP}}_C - \mathsf{MP}_C| \leq \zeta := \left(1 + \frac{\xi}{p_{\mathsf{min}}}\right)^{2 \cdot |C|} - 1 \,. \tag{3.2}$$

Proof. Consider a Markov chain C with a reward function $r : S \to [0, 1]$, such that C is a single BSCC. The discounted sum MD^λ for a state s of C is defined as:

$$\mathsf{MD}^\lambda(s) := \lim_{n \to \infty} \mathbb{E}\left(\frac{\sum_{i=1}^{n} r(S_{i,s})\lambda^i}{\sum_{i=1}^{n} \lambda^i}\right),$$

where $\lambda > 0$ is a discount factor and $S_{i,s}$ is a random variable that returns the i-th state on a run that starts in the state s. We say that a Markov chain $\hat{C}$ is *ξ-close to C* if

1. $\hat{C}$ is over the same states as C,
2. $\forall s, s' \in C : |\mathbf{P}_C(s, s') - \mathbf{P}_{\hat{C}}(s, s')| \leq \xi$,
3. $\forall s, s' \in C : \mathbf{P}_C(s, s') > 0 \iff \mathbf{P}_{\hat{C}}(s, s') > 0$.

We write $\hat{\mathsf{MD}}^\lambda$ for the discounted sum computed for $\hat{C}$. By [69](Theorem 4) it holds that for every discount factor $0 < \lambda < 1$, every MC $\hat{C}$ that is ξ-close to C, and every state s:

$$|\hat{\mathsf{MD}}^\lambda(s) - \mathsf{MD}^\lambda(s)| \leq \left(1 + \frac{\xi}{p_{\mathsf{min}}}\right)^{2 \cdot |C|} - 1 \,, \tag{3.3}$$

where p_{min} is the minimum transition probability in $\mathcal{M}$. By [70] we know that the discounted sum converges to mean payoff:

$$\lim_{\lambda \to 1} \mathsf{MD}^\lambda(s) = \mathsf{MP}_C \qquad\qquad \lim_{\lambda \to 1} \hat{\mathsf{MD}}^\lambda(s) = \hat{\mathsf{MP}}_C,$$

where MP_C and $\hat{\mathsf{MP}}_C$ are the mean payoff for C and $\hat{C}$, respectively. We obtain the result by taking the limit $\lambda \to 1$ in (3.3). □

Note that by Taylor's expansion, for small ξ, we have $\zeta \approx 2|C|\xi$.

Algorithm 3.4 SINGLEPATHMP

Input: reward function r, $p_{\mathsf{min}}, \zeta, \delta \in (0, 1]$,

Output: $\hat{\mathsf{MP}}_C$ such that $|\hat{\mathsf{MP}}_C - \mathsf{MP}_C| < \zeta$ where C is the BSCC of the generated run

$\pi \leftarrow \lambda$

repeat

$\pi \leftarrow \pi \,.\, \mathsf{NextState}(\pi)$

if $K(\pi) \neq \bot$ **then**

$\xi = p_{\mathsf{min}}((1+\zeta)^{1/2|K(\pi)|} - 1)$ ▷ By Equation (3.2)

$k \leftarrow \frac{\ln(2|K(\pi)|^2) - \ln(\delta/2)}{2\xi^2}$ ▷ By Equation (3.1)

until REACHEDBSCC$(\pi, p_{\mathsf{min}}, \delta/2)$ and SCAND$_k(K(\pi), \pi)$

return $\hat{\mathsf{MP}}_{K(\pi)}$ computed from $\hat{\mathbf{P}}$ and r

Sampling algorithm Algorithm 3.4 extends Algorithm 3.2 as follows. It divides the confidence parameters δ into δ_{BSCC} (used as in Algorithm 3.2 to detect the BSCC) and δ_C (the total confidence for the estimates on transition probabilities). For simplicity, we set $\delta_{BSCC} = \delta_C = \delta/2$. First, we compute the bound ξ required for ζ-precision (by (3.2)). Subsequently, we compute the required strength k of the candidate guaranteeing δ_C-confidence on $\hat{\mathbf{P}}$ (from (3.1)). The path is prolonged until the candidate is strong enough; in such a case $\hat{\mathsf{MP}}_C$ is ζ-approximated with $1 - \delta_C$ confidence. If the candidate of the path changes, all values are computed from scratch for the new candidate.

Theorem 3.5. *For every $\delta > 0$, the Algorithm 3.4 terminates correctly with probability at least $1 - \delta$.*

Proof. From (3.1), by the union bound, we are guaranteed that the probability that *none* of the estimates $\hat{\mathbf{P}}_{s,s'}$ is outside of the ζ-neighbourhood doesn't exceed the sum of all respective estimation errors, that is, $\delta_C = \sum_{s,s' \in C} \delta_{s,s'}$. Next, from (3.2) and from the fact that C is subject to Theorem 3.1 with confidence δ_{BSCC},

$$
\begin{aligned}
&P(|\mathsf{MP}_C(r) - \hat{\mathsf{MP}}_C(r)| > \zeta) = \\
&\quad = P(C \in \mathsf{BSCC}) P(|\mathsf{MP}(r) - \hat{\mathsf{MP}}(r)| > \zeta \mid C \in \mathsf{BSCC}) + \\
&\quad\quad P(C \notin \mathsf{BSCC}) P(|\mathsf{MP}(r) - \hat{\mathsf{MP}}(r)| > \zeta \mid C \notin \mathsf{BSCC}) \\
&\quad \leq 1 \cdot \delta_C + \delta_{BSCC} \cdot 1 = \delta_C + \delta_{BSCC} \leq \delta.
\end{aligned}
$$

□

3.5.1 Hypothesis testing with bounded error

Let random variable $X_{\mathsf{MP}}^{\zeta,\delta}$ denote the value SINGLEPATHMP$(r, p_{\mathsf{min}}, \zeta, \delta)$. The following theorem establishes relation between the mean-payoff MP and the expected value of $X_{\mathsf{MP}}^{\zeta,\delta}$.

Theorem 3.6. *For every* $\delta, \zeta > 0$, $\mathsf{MP} - \zeta - \delta \leq \mathbb{E}(X_{\mathsf{MP}}^{\zeta,\delta}) \leq \mathsf{MP} + \zeta + \delta$.

Proof. Let us write $X_{\mathsf{MP}}^{\zeta,\delta}$ as an expression of random variables Y, W, Z

$$X_{\mathsf{MP}}^{\zeta,\delta} \;=\; Y(1-W) + WZ,$$

where (i) W is a Bernoulli random variable, such that $W = 0$ iff the algorithm correctly detected the BSCC and estimated transition probabilities within bounds, (ii) Y is the value computed by the algorithm if $W = 0$, and the real mean payoff MP when $W = 1$, and (iii) Z is any random variable with the range $[0, 1]$. The interpretation is as follows: when $W = 0$ we observe the result Y, which has bounded error ζ, and when $W = 1$ we observe arbitrary Z. We note that Y, W, Z are not necessarily independent. By Theorem 3.5 $\mathbb{E}(W) \leq \delta$ and by linearity of expectation: $\mathbb{E}(X_{\mathsf{MP}}^{\zeta,\delta}) = \mathbb{E}(Y) - \mathbb{E}(YW) + \mathbb{E}(WZ)$. For the upper bound, observe that $\mathbb{E}(Y) \leq \mathsf{MP} + \zeta$, $\mathbb{E}(YW)$ is non-negative and $\mathbb{E}(WZ) \leq \delta$. As for the lower bound, note that $\mathbb{E}(Y) \geq \mathsf{MP} - \zeta$, $\mathbb{E}(YW) \leq \delta$ and $\mathbb{E}(WZ)$ is non-negative. □

As a consequence of Theorem 3.6, if we establish that with $(1-\alpha)$ confidence $X_{\mathsf{MP}}^{\zeta,\delta}$ belongs to the interval $[a, b]$, then we can conclude with $(1-\alpha)$ confidence that MP belongs to the interval $[a - \zeta - \delta, b + \zeta + \delta]$. Standard statistical methods can be applied to find the confidence bound for $X_{\mathsf{MP}}^{\zeta,\delta}$; for instance the method presented in Section 2.3.2.

3.6 Experimental Evaluation

We implemented our algorithms in the probabilistic model checker PRISM [62], and evaluated them on the DTMC examples from the PRISM benchmark suite [64]. The benchmarks model communication and security protocols, distributed algorithms, and fault-tolerant systems. To demonstrate how our method performs depending on the topology of Markov chains, we also

performed experiments on the generic DTMCs shown in Figure 3.3 and Figure 3.4, as well as on two CTMCs from the literature that have large BSCCs: “tandem” [71] and “gridworld” [58].

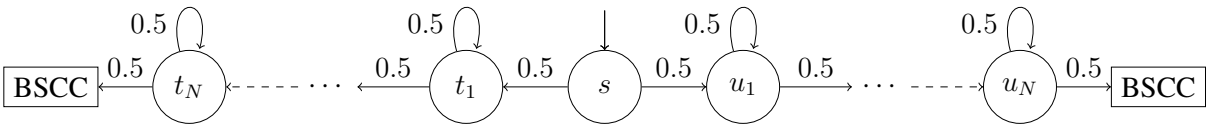

Figure 3.4: A Markov chain with two transient parts consisting of N strongly connected singletons, leading to BSCCs with the ring topology of M states.

All benchmarks are parametrised by one or more values, which influence their size and complexity, e.g. the number of modelled components. We have made minor modifications to the benchmarks that could not be handled directly by the SMC component of PRISM, by adding self-loops to deadlock states and fixing one initial state instead of multiple.

Experiments were done on a Linux 64-bit machine running an AMD Opteron 6134 CPU with a time limit of 15 minutes and a memory limit of 5GB. To increase performance of our tool, we check whether a candidate has been found every 1000 steps; this optimization does not violate correctness of our analysis.

Reachability The experimental results for unbounded reachability are shown in Tables 3.2 and 3.3. The PRISM benchmarks were checked against their standard properties, when available. We directly compare our method to another topology-agnostic method of [30] (SimTermination), where at every step the sampled path is terminated with probability p_{term}. The approach of [31] with a priori bounds is not included, since it times out even on the smallest benchmarks. In addition, we performed experiments on two methods that are topology-aware: sampling with reachability analysis of [30] (SimAnalysis) and the numerical model-checking algorithm of PRISM (MC).

The table shows the size of every example, its minimum probability, the number of BSCCs, and the size of the largest BSCC. Column “time” reports the total wall time for the respective algorithm, and “analysis” shows the time for symbolic reachability analysis in the SimAnalysis method. Highlights show the best result among the topology-agnostic methods. All statistical methods were used with the SPRT test for choosing between the hypothesis (see Section 2.3.1), and their results were averaged over five runs.

Example			BSCC	SimAdaptive	SimTermination[30]	SimAnalysis[30]		MC
name	size	p_{min}	no., max. size	time	time	time	analysis	time
bluetooth(4)	149K	$7.8 \cdot 10^{-3}$	3K, 1	2.6s	16.4s	83.2s	80.4s	78.2s
bluetooth(7)	569K	$7.8 \cdot 10^{-3}$	5.8K, 1	3.8s	50.2s	284.4s	281.1s	261.2s
bluetooth(10)	>569K	$7.8 \cdot 10^{-3}$	>5.8K, 1	5.0s	109.2s	TO	-	TO
brp(500,500)	4.5M	0.01	1.5K, 1	7.6s	13.8s	35.6s	30.7s	103.0s
brp(2K,2K)	40M	0.01	4.5K, 1	20.4s	17.2s	824.4s	789.9s	TO
brp(10K,10K)	>40M	0.01	>4.5K, 1	89.2s	15.8s	TO	-	TO
crowds(6,15)	7.3M	0.066	>3K, 1	3.6s	253.2s	2.0s	0.7s	19.4s
crowds(7,20)	17M	0.05	>3K, 1	4.0s	283.8s	2.6s	1.1s	347.8s
crowds(8,20)	68M	0.05	>3K, 1	5.6s	340.0s	4.0s	1.9s	TO
eql(15,10)	616G	0.5	1, 1	16.2s	TO	151.8s	145.1s	110.4s
eql(20,15)	1279T	0.5	1, 1	28.8s	TO	762.6s	745.4s	606.6s
eql(20,20)	1719T	0.5	1, 1	31.4s	TO	TO	-	TO
herman(17)	129M	$7.6 \cdot 10^{-6}$	1, 34	23.0s	33.6s	21.6s	0.1s	1.2s
herman(19)	1162M	$1.9 \cdot 10^{-6}$	1, 38	96.8s	134.0s	86.2s	0.1s	1.2s
herman(21)	10G	$4.7 \cdot 10^{-7}$	1, 42	570.0s	TO	505.2s	0.1s	1.4s

Table 3.2: Experimental results for unbounded reachability; part 1/2. Simulation parameters: $\alpha = \beta = \varepsilon = 0.01$, $\delta = 0.001$, $p_{\mathsf{term}} = 0.0001$. TO means time-out, and MO means memory-out. Our approach is denoted by SimAdaptive here. Highlights show the best result the among topology-agnostic methods.

Finding the optimal termination probability p_{term} for the SimTermination method is a non-trivial task. If the probability is too high, the method might never reach the target states, thus give an incorrect result, and if the value is too low, then it might sample unnecessarily long traces that never reach the target. For instance, to ensure a correct answer on the Markov chain in Figure 3.3, p_{term} has to decrease exponentially with the number of states. By experimenting we found that the probability $p_{\mathsf{term}} = 0.0001$ is low enough to ensure correct results.

On most examples our method scales better than the SimTermination method. Our method performs well even on examples with large BSCCs, such as "tandem" and "gridworld," due to early termination when a goal state is reached. For instance, on the "gridworld" example, most BSCCs do not contain a goal state, thus have to be fully explored, however the probability of reaching such BSCC is low, and as a consequence full BSCC exploration rarely occurs. The SimTermination method performs well when the target states are unreachable or can be

Example			BSCC	SimAdaptive	SimTermination[30]	SimAnalysis[30]		MC
name	size	$p_{\min}$	no., max. size	time	time	time	analysis	time
leader(6,6)	280K	$2.1 \cdot 10^{-5}$	1, 1	5.0s	5.4s	536.6s	530.3s	491.4s
leader(6,8)	>280K	$3.8 \cdot 10^{-6}$	1, 1	23.0s	26.0s	MO	-	MO
leader(6,11)	>280K	$5.6 \cdot 10^{-7}$	1, 1	153.0s	174.8s	MO	-	MO
nand(50,3)	11M	0.02	51, 1	7.0s	231.2s	36.2s	31.0s	272.0s
nand(60,4)	29M	0.02	61, 1	6.0s	275.2s	60.2s	56.3s	TO
nand(70,5)	67M	0.02	71, 1	6.8s	370.2s	148.2s	144.2s	TO
tandem(500)	>1.7M	$2.4 \cdot 10^{-5}$	1, >501K	2.4s	6.4s	4.6s	3.0s	3.4s
tandem(1K)	1.7M	$9.9 \cdot 10^{-5}$	1, 501K	2.6s	19.2s	17.0s	12.7s	13.0s
tandem(2K)	>1.7M	$4.9 \cdot 10^{-5}$	1, >501K	3.4s	72.4s	62.4s	59.8s	59.4s
gridworld(300)	162M	$1 \cdot 10^{-3}$	598, 89K	8.2s	81.6s	MO	-	MO
gridworld(400)	384M	$1 \cdot 10^{-3}$	798, 160K	8.4s	100.6s	MO	-	MO
gridworld(500)	750M	$1 \cdot 10^{-3}$	998, 250K	5.8s	109.4s	MO	-	MO
Fig.3.3(16)	37	0.5	1, 1	58.6s	TO	23.4s	0.4s	2.0s
Fig.3.3(18)	39	0.5	1, 1	TO	TO	74.8.0s	1.8s	2.0s
Fig.3.3(20)	41	0.5	1, 1	TO	TO	513.6s	11.3s	2.0s
Fig.3.4(1K,5)	4022	0.5	2, 5	7.8s	218.2s	3.2s	0.5s	1.2s
Fig.3.4(1K,50)	4202	0.5	2, 50	12.4s	211.8s	3.6s	0.7s	1.0s
Fig.3.4(1K,500)	6002	0.5	2, 500,	431.0s	218.6s	3.6s	1.0s	1.2s
Fig.3.4(10K,5)	40K	0.5	2, 5	52.2s	TO	42.2s	25.4s	25.6s
Fig.3.4(100K,5)	400K	0.5	2, 5	604.2s	5.4s	TO	-	TO

Table 3.3: Experimental results for unbounded reachability; part 2/2. Simulation parameters are as in Table 3.2.

reached by short paths. When long paths are necessary to reach the target, the probability that an individual path reaches the target is small, hence many samples are necessary to estimate the real probability with high confidence.

Moreover, it turns out that our method compares well even with methods that have access to the topology of the system. In many cases, the running time of the numerical algorithm MC increases dramatically with the size of the system, while remaining almost constant in our method. The bottleneck of the SimAnalysis algorithm is the reachability analysis of states that cannot reach the target, which in practice can be as difficult as numerical model checking.

LTL **and mean payoff** In the second experiment, we compared our algorithm for checking LTL properties and estimating the mean payoff with the numerical methods of PRISM; the results

are shown in Table 3.4 and 3.5. We compare against PRISM, since we are not aware of any SMC-based or topology-agnostic approach for mean payoff, or full LTL. For mean payoff, we computed 95%-confidence bound of size 0.22 with parameters $\delta = 0.011, \zeta = 0.08$, and for LTL we used the same parameters as for reachability. We report results only on a single model of each type, where either method did not time out. In general our method scales better when BSCCs are fairly small and are discovered quickly.

Example		LTL	
name	property	SimAdaptive time	MC time
bluetooth(10)	$\Box\Diamond$	8.0s	TO
brp(10K,10K)	$\Diamond\Box$	90.0s	TO
crowds(8,20)	$\Diamond\Box$	9.0s	TO
eql(20,20)	$\Box\Diamond$	7.0s	MO
herman(21)	$\Box\Diamond$	TO	2.0s
leader(6,5)	$\Box\Diamond$	277.0s	117.0s
nand(70,5)	$\Box\Diamond$	4.0s	TO
tandem(2K)	$\Box\Diamond$	TO	221.0s
gridworld(100)	$\Box\Diamond \rightarrow \Diamond\Box$	TO	110.4s
Fig.3.3(20)	$\Box\Diamond \rightarrow \Box\Diamond$	TO	
Fig.3.4(100K,5)	$\Box\Diamond$	348.0s	TO
Fig.3.4(1K,500)	$\Box\Diamond$	827.0s	2.0s

Table 3.4: Experiment results for LTL. The following simulation parameters were used: $\alpha = \beta = \varepsilon = 0.01$, $\delta = 0.001$.

3.7 Theoretical Bounds

As demonstrated by the experimental results, our method is fast on systems that are (i) shallow, and (ii) with small BSCCs. In such systems, the BSCC is reached quickly and the candidate is built-up quickly. Further, recall that the BSCC is reported when a k-candidate is found, and that k is increased with each candidate along the path. Hence, when there are many strongly connected sets, and thus many candidates, the BSCC is detected by a k-candidate for a large k. However, since k grows linearly in the number of candidates, the most important and limiting factor is the size of BSCCs.

Example name	Mean payoff SimAdaptive time	MC time
bluetooth(10)	3.0s	TO
brp(10K,10K)	6.6s	TO
crowds(8,20)	2.0s	TO
eql(20,20)	2.6s	TO
herman(21)	MO	3.0s
leader(6,6)	48.5	576.0
nand(70,5)	2.0s	294.0s
tandem(500)	TO	191.0s
gridworld(50)	TO	58.1s
Fig.3.3(20)	TO	1.8s
Fig.3.4(100K,5)	79.6s	TO
Fig.3.4(1K,500)	TO	2.0s

Table 3.5: Experimental results for mean-payoff properties. For mean-payoff we computed a 95%-confidence interval of size 0.22 with $\delta = 0.011, \zeta = 0.08$.

Theoretical bound We state the dependency on the depth of the system and BSCC sizes formally. We pick $\delta = \frac{\varepsilon}{2}$ and let

$$sim = \frac{-\log\frac{\beta}{1-\alpha}\log\frac{1-\beta}{\alpha}}{\log\frac{p-\varepsilon+\delta}{p+\varepsilon-\delta}\log\frac{1-p-\varepsilon+\delta}{1-p+\varepsilon-\delta}} \qquad \text{and} \qquad k_i = \frac{i - \log\delta}{-\log(1-p_{\mathsf{min}})}$$

denote the a priori upper bound on the number of simulations necessary for the SPRT and the strength of candidates as in Algorithm 3.2, respectively.

Theorem 3.7. *Let R denote the expected number of steps before reaching a BSCC and B the maximum size of a BSCC. Further, let*

$$T = \max_{C \in \mathsf{BSCC}; s,s' \in C} \mathbb{E}(\textit{time to reach } s' \textit{ from } s).$$

In particular, $T \in \mathcal{O}(B/p_{\mathsf{min}}^B)$. Then the expected running time of Algorithms 3.2 and 3.3 is at most

$$\mathcal{O}(sim \cdot k_{R+B} \cdot B \cdot T)\,.$$

Proof. We show that the expected running time of each simulation is at most $k_{R+B} \cdot B \cdot T$. Since the expected number of states visited is bounded by $R + B$, the expected number of candidates

on a run is less than $2(R+B)-1$. Since k_i grows linearly in i it is sufficient to prove that the expected time to visit each state of a BSCC once (when starting in BSCC) is at most $B \cdot T$. We order the states of a BSCC as $s_1, \ldots, s_b$, then the time is at most $\sum_{i=1}^{b} T$, where $b \leq B$. This yields the result since $R \in \mathcal{O}(k_{R+B} \cdot B \cdot T)$.

It remains to prove that $T \leq B/p_{\mathsf{min}}^B$. Let s be a state of a BSCC of size at most B. Then, for any state s' from the same BSCC, the shortest path from s to s' has length at most B and probability at least p_{min}^B. Consequently, if starting at s, we haven't reached s' after B steps with probability at most $1 - p_{\mathsf{min}}^B$, and we are instead in some state $s'' \neq s'$, from which, again, the probability to reach s' within B steps at least p_{min}^B. Hence, the expected time to reach s' from s is at most

$$\sum_{i=1}^{\infty} B \cdot i(1 - p_{\mathsf{min}}^B)^{i-1} p_{\mathsf{min}}^B,$$

where i indicates the number of times a sequence of B steps is observed. The series can be summed by differentiating a geometric series. As a result, we obtain a bound B/p^B. □

Systems that have large deep BSCCs require longer time to reach for the required level of confidence. However, such systems are often difficult to handle also for other methods agnostic of the topology. For instance, correctness of [30] on the example in Figure 3.3 relies on the termination probability p_{term} being at most $1 - \lambda$, which is less than 2^{-n} here. Larger values lead to incorrect results and smaller values to paths of exponential length. Nevertheless, our procedure usually runs faster than the bound suggest.

Theoretical vs. empirical running time We now compare the theoretical upper bound on running time given in Theorem 3.7 to empirical data. We omit the number of simulation runs (term sim in the theorem), and report only the logarithm of the average simulation length. Figures 3.5, 3.6 and 3.7 present the comparison for different topologies of Markov chains. In Figure 3.5 we present the comparison for the worst-case Markov chain, which requires the longest paths to discover the BSCCs as a k-candidate. This Markov chain is like the one in Figure 3.3, but where the last state has a single outgoing transition to the initial state. Figure 3.6 suggests that the theoretical bound can be a good predictor of running time with respect to the depth of the system, however, Figure 3.7 shows that it is very conservative with respect to the size of BSCCs.

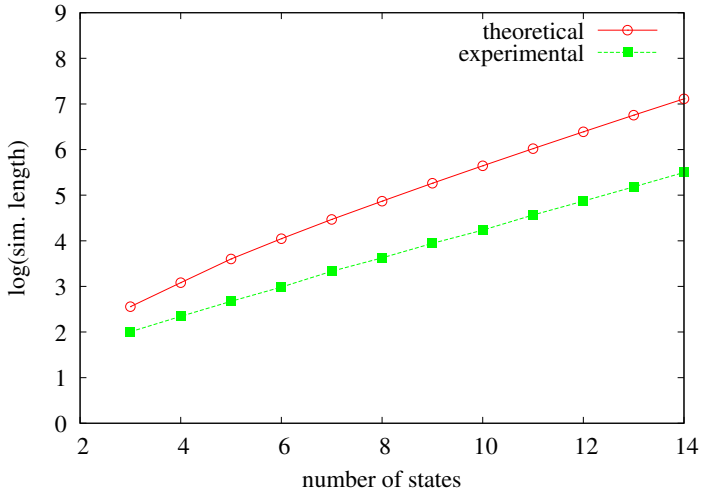

Figure 3.5: Average length of simulations for a Markov chain like in Figure 3.3, but where the last state has a single outgoing transition to the initial state.

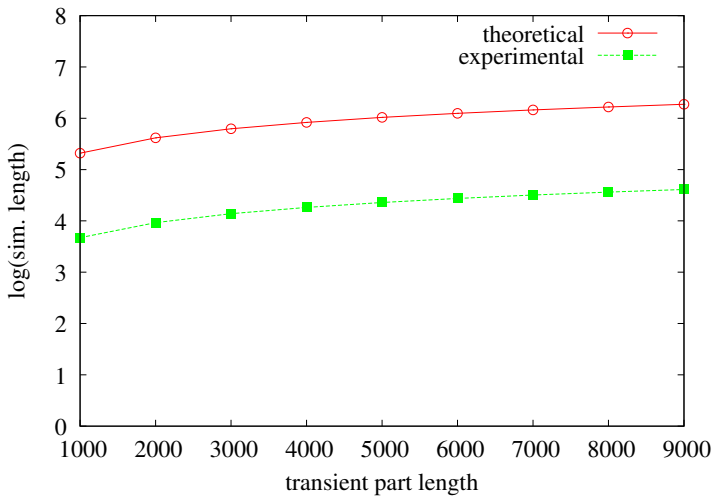

Figure 3.6: Average length of simulations for the MC in Figure 3.4, where $M = 5$ and N varies.

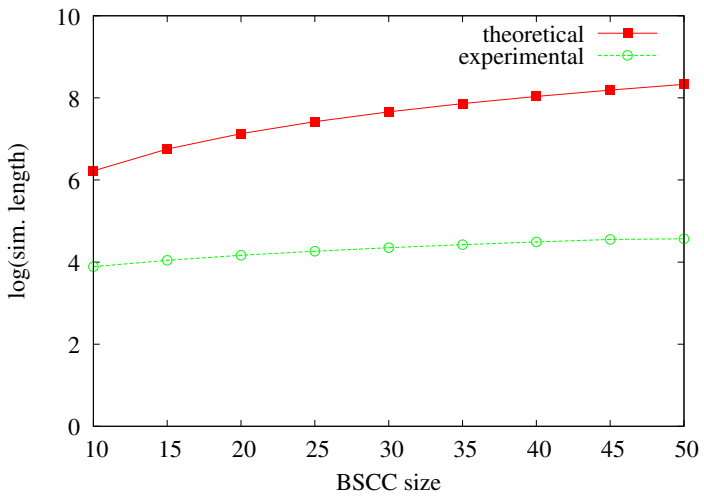

Figure 3.7: Average length of simulations for the MC in Figure 3.4, where $N = 1000$ and M varies.

4 Linear Distances between Markov Chains

4.1 Introduction

Behavior of processes is traditionally compared using various notions of equivalence, such as trace equivalence, bisimulation, etc. However, the concept of equivalence is often too coarse for quantitative systems, such as Markov chains. For instance, probabilities of failures of particular hardware components are typically only empirically estimated and the slightest imprecision in the estimate may result in breaking the equivalence between processes. Moreover, if the (possibly black-box) processes are indeed different we would like to measure *how much they differ*. This has led to lifting the Boolean idea of behavioral equivalence to a finer, quantitative notion of behavioral *distance* between processes. The distance between processes s and t is typically formalized as $\sup_{p \in \mathcal{C}} |p(s) - p(t)|$ where $\mathcal{C}$ is a class of properties of interest and $p(s)$ is a quantitative value of the property p in process s [72]. This notion has been introduced in [72] for Markov chains and further developed in various settings, such as Markov decision processes [73], quantitative transition systems [74], or concurrent games [75].

Several kinds of distances have been investigated for Markov chains. On the one hand, *branching* distances, e.g. [76; 72; 77; 78; 79; 80; 81; 82], lift the equivalence given by the probabilistic bisimulation of Larsen and Skou [20]. On the other hand, there are *linear* distances, in particular the total variation distance [83; 84] and trace distances [85; 86]. Linear distances are particularly appropriate when (i) we are interested in linear-time properties, and (ii) we want to estimate the distance based only on simulation runs from the initial distribution of the system, i.e. in a black-box setting. (Recall that for branching distances, the underlying probabilistic

bisimulation corresponds to testing equivalence where not only runs from the initial distribution can be observed, but it is also possible to dump the current state of the system, and later restart the simulation from this state [20].)

In this chapter, we introduce a simple framework for linear distances between Markov chains, using the formula above, where $p(s)$ is the probability of satisfying p when starting a simulation run in state s (when p is seen as a language of ω-words it is the probability to generate a trace belonging to p). We consider several classes $\mathcal{C}$ of languages of interest, characterized from several points of view, e.g. topologically, by linear-time logics, or by automata, thus rendering our framework versatile.

We investigate when a given distance can be estimated in a black-box setting, i.e. only from simulations. One of the main difficulties is that the class $\mathcal{C}$ typically includes properties with arbitrarily long horizon or even infinite-horizon properties, whereas every simulation run is necessarily finite. Note that we do not employ any simplifications such as imposed fixed horizon or discounting, typically used for obtaining efficient algorithms, e.g., [72; 77; 80], and the undiscounted setting is fundamentally more complex [78]. Since even simpler tasks are impossible for unbounded horizon in the black-box setting without any further knowledge, we assume we only know a lower bound on the minimum transition probability p_{min}. The knowledge of p_{min} has been already justified in Chapter 3.

Our contribution are the following:

- We introduce a systematic linear-distance framework and illustrate it with several examples, including distances previously investigated in the literature.

- The main technical contributions are (i) a negative result stating that the total variation distance cannot be estimated by simulating the systems, and (ii) a positive result that the trace distance can be estimated.

- These results are further exploited to provide both negative and positive results for each of the settings where the language class is given topologically, by LTL (linear temporal logic) fragments, and by automata. We also show that the negative result on the total variation distance can be turned into a positive result if the transition probabilities have finite precision.

Outline In Section 4.2 we review the related work. We introduce our framework and illustrate it with examples in Section 4.3. We define our problem formally in Section 4.3.2. In Sections 4.4 and 4.5 we provide the proofs of our technically principal negative and positive result, respectively. Section 4.6 extends the results in the settings of topology, logics and automata, and discusses general conditions for estimability.

4.2 Related Work

There are two main linear distances considered for Markov chains: the total variation distance and trace distance. Several algorithms have been proposed for both of them in the case when the Markov chains are known (the white-box setting). We are not aware of any work where the distances are estimated only from simulating the systems (the black-box setting).

Firstly, for the *total variation distance* in the white-box setting, [83] shows that deciding whether it equals one can be done in polynomial time, but computing it is $\mathbf{NP}$-hard and not known to be decidable, however, it can be approximated; [84] considers this distance more generally for semi-Markov processes, provides a different approximation algorithm, and shows it coincides with distances based on (i) metric temporal logic, and (ii) timed automata languages.

Secondly, the *trace distance* is based on the notion of trace equivalence, which can be decided in polynomial time [87] (however, trace refinement of Markov decision processes is already undecidable [88]). Several variants of trace distance are considered in [85] where it is taken as a limit of finite-trace distances, possibly using discounting or averaging. In [86] the finite-trace distance is shown to coincide with distances based on (i) LTL, and (ii) LTL without the $\mathcal{U}$ operator, i.e., only using the $\bigcirc$ operator and Boolean connectives. This distances is also shown to be $\mathbf{NP}$-hard and not known to be decidable, similarly to the total variation distance. Finally, an approximation algorithm is shown (again in the white-box setting), where the over-approximates are branching-time distances, showing an interesting connection between the branching and linear distances.

In [89] the distinguishability problem is considered, i.e. given two Markov chains whether there is a monitor that reads a single sample and with high probability decides which chain produced the sequence. This is indeed possible when the total variation distance between the chains equals one, and [89] shows how to construct such monitors. In contrast, our negative

results shows that it is not possible to decide with high probability whether the total variation distance equals one when the two Markov are black-box.

Linear distances have been proposed also for quantitative transition systems, e.g. [75]. Moreover, there are other useful distances based on different fundaments; for instance, the Skorokhod distance [90; 91; 92] measures the discrete differences between systems while allowing for timing distortion; Kullback-Leibler divergence [85] is useful from the information-theoretic point of view. Finally, distances have been also studied with respect to applications in linear-time model checking [93; 86].

4.3 Framework for Linear Distances

In this section we introduce our framework for linear distances. For $i \in \{1, 2\}$, let $\mathcal{M}_i = (S, \mathbf{P}_i, L, \mu_i)$ denote a Markov chain and $(\mathsf{Runs}, \mathcal{F}, \mathbb{P}_i)$ the induced probability space (see Section 2.2 for definitions). To avoid clutter, the chains are defined over the same state space with the same labelling, which can be w.l.o.g. achieved by their disjoint union. The measurable space of ω-languages is given by the set $(2^{Ap})^\omega$ equipped with a σ-algebra $\mathcal{F}(2^{Ap})$ generated by the set of cones $\{\overline{w}(2^{Ap})^\omega \mid \overline{w} \in (2^{Ap})^*\}$. This ensures, for every measurable ω-language X, that $L^{-1}(X)$ is measurable in every Markov chain.

Since single runs of Markov chains typically have measure 0, we introduce linear distances using measurable sets of runs:

Definition 4.1 ($\mathcal{L}$-distance)**.** *For a class $\mathcal{L} \subseteq \mathcal{F}$ of measurable ω-languages, the $\mathcal{L}$-distance $\mathsf{D}_\mathcal{L}$ is defined by*

$$\mathsf{D}_\mathcal{L}(\mathcal{M}_1, \mathcal{M}_2) = \sup_{X \in \mathcal{L}} |\mathbb{P}_1(X) - \mathbb{P}_2(X)|.$$

Note that every $\mathsf{D}_\mathcal{L}$ is a pseudo-metric, i.e. it is symmetric, it satisfies the triangle inequality, and the distance between identical Markov chains is 0. However, two different Markov chains can have distance 0, for instance, when they induce the same probability space.

4.3.1 Examples of Linear Distances

The definition of $\mathcal{L}$-distances can be instantiated either (i) by a direct topological description of $\mathcal{L}$, or indirectly (ii) by a class $\mathbb{A}$ of automata inducing the class of recognized languages $\mathcal{L} = \{\mathcal{L}(\mathcal{A}) \mid \mathcal{A} \in \mathbb{A}\}$, or (iii) by a set of formulae $\mathfrak{L}$ of a linear-time logic inducing the languages of models $\mathcal{L} = \{\mathcal{L}(\varphi) \mid \varphi \in \mathfrak{L}\}$ where $\mathcal{L}(\varphi)$ denotes the language of ω-words satisfying the formula φ.

We now discuss several particularly interesting instantiations:

Total variation One extreme choice is to consider all measurable languages, resulting in the *total variation distance* $\mathsf{D}_{\mathsf{TV}}(\mathcal{M}_1, \mathcal{M}_2) = \sup_{X \in \mathcal{F}(2^{Ap})} |\mathbb{P}_1(X) - \mathbb{P}_2(X)|$.

Trace distances The other extreme choices are to consider (i) only the generators of $\mathcal{F}(2^{Ap})$, i.e. the cones $\{w(2^{Ap})^\omega \mid \overline{w} \in (2^{Ap})^*\}$, resulting in the *finite-trace distance* $\mathsf{D}_{\mathsf{FT}}(\mathcal{M}_1, \mathcal{M}_2) = \sup_{w \in (2^{Ap})^+} |\mathbb{P}_1(w) - \mathbb{P}_2(w)|$; or (ii) only the elementary events, i.e. $(2^{Ap})^\omega$, resulting in the *infinite-trace distance* $\mathsf{D}_{\mathsf{IT}}(\mathcal{M}_1, \mathcal{M}_2) = \sup_{w \in (2^{Ap})^\omega} |\mathbb{P}_1(w) - \mathbb{P}_2(w)|$.

Topological distances There are many possible choices for $\mathcal{L}$ between the two extremes above, such as *clopen sets* Δ_1, which are finite unions of cones (being both closed and open), *open sets* 2^{Ap}_1, which are infinite unions of cones, *closed sets* Π_1, or classes higher in the *Borel hierarchy* such as the class of ω-*regular* languages (within Δ_3), or languages given by thresholds for a *long-run average reward* (within 2^{Ap}_3).

Automata distances The class of ω-regular languages can also be given in terms of automata, for instance by the class of all deterministic Rabin automata (DRA). Similarly, the closed sets Π_1 correspond to the class of deterministic Büchi automata with all states final. Further, we can restrict the class of all DRA to those of *size at most* k for a fixed $k \in \mathbb{N}$, denoting the resulting distance by $\mathsf{D}_{DRA \leq k}$.

Logical distances The class of ω-regular languages can also be given in terms of logic, by the monadic second-order logic (with order). Further useful choices include *first-order logic with order*, corresponding to the star-free languages and to *linear temporal logic* (LTL), or its fragments such as LTL with only $\bigcirc$ or only $\Diamond$ and $\Box$ operators etc.

Finite-trace equivalence The introduced distances can also be considered in the discrete setting, resulting in various notions of equivalence. For instance, the *finite-trace equivalence* E_{FT} can be derived from the finite-trace distance by the following discretization:

$$\mathsf{E}_{\mathsf{FT}}(\mathcal{M}_1, \mathcal{M}_2) = \begin{cases} 0 & \text{if } \mathsf{D}_{\mathsf{FT}}(\mathcal{M}_1, \mathcal{M}_2) = 0 \\ 1 & \text{otherwise, i.e., } \mathsf{D}_{\mathsf{FT}}(\mathcal{M}_1, \mathcal{M}_2) > 0. \end{cases}$$

4.3.2 Problem Statement

Linear distances can be very useful when we want to compare a black-box system with another system, e.g. a white-box specification or a black-box previous version of the system. Indeed, in such a setting we can typically obtain simulation runs of the system and we must establish a relation between the systems based on these runs only. This is in contrast with branching distances where either both systems are assumed white-box or there are strong requirements on the testing abilities, such as dumping the current state of the system, arbitrary many restarts from there, and nesting this branching arbitrarily. Therefore, we focus on the setting where we can obtain only finite prefixes of runs and we use statistics to (i) deduce information on the whole infinite runs, and (ii) estimate the distance of the systems.

Definition 4.2 (Estimability)**.** *A distance function* $\mathsf{D}_{\mathcal{L}}$ *is called* estimable*, if there exists an almost-surely terminating algorithm that given*

- *any desired finite number of sampled simulation run from Markov chains* $\mathcal{M}_1$ *and* $\mathcal{M}_2$ *of any desired finite length,*
- *lower bound* $p_{\mathsf{min}} > 0$ *on the minimum (non-zero) transition probability,*
- *confidence* $\alpha \in (0, 1)$*,*
- *interval width* $\delta \in (0, 1)$*,*

computes an interval I *such that* $|I| \leq \delta$ *and*

$$\mathbb{P}[\mathsf{D}_{\mathcal{L}}(\mathcal{M}_1, \mathcal{M}_2) \in I] \geq 1 - \alpha.$$

A distance function is called inestimable*, if there does not exists an algorithm in the above sense.*

4.4 Inestimability: Total Variation Distance

We show that for the total variation distance D_{TV} there exists no "statistical" algorithm (in the above sense) which is correct for all inputs $(\mathcal{M}_1, \mathcal{M}_2, \alpha, \delta)$. Our argument consists of two steps:

1. We construct two chains such that $\mathsf{D}_{\mathsf{TV}}(\mathcal{M}_1, \mathcal{M}_2) = 1$, namely the two Markov chains shown in Figure 4.1 (similar to [85]): one with $\tau = 0$ and the other with small $\tau > 0$.

2. We show that any potentially correct algorithm will give with high probability an incorrect output for some choice of τ, α, δ.

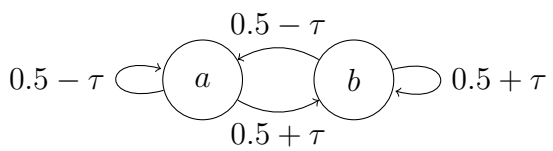

Figure 4.1: A Markov chain with labelling displayed in states.

4.4.1 Maximizing Event

We start by showing that even an arbitrarily small difference in transition probabilities between two Markov chains may result in total variation distance of 1. Consider the two Markov chains as in Figure 4.1, where $\mathcal{M}_1$ has $\tau = 0$, and $\mathcal{M}_2$ has $\tau > 0$. We assume that the initial distribution for each chain is its stationary distribution. In this setting, every simulation step is like an independent trial with probability $0.5 - \tau$ (resp. $0.5 + \tau$) of seeing a (resp. b).

Let X_n (resp. Y_n) denote the number of b symbols in a random path of length n sampled from $\mathcal{M}_1$ (resp. $\mathcal{M}_2$). By the central limit theorem the distributions of X_n and Y_n are converging to the normal distribution when $n \to \infty$:

$$X_n \approx \mathcal{N}(0.5n, 0.5^2 n) \qquad\qquad Y_n \approx \mathcal{N}((0.5+\tau)n, n(0.25-\tau^2)).$$

For $n \in \mathbb{N}$ let the event E_n mean "there is at most $c_n = (0.5 + \tau/2)n$ symbols b in the path prefix of length n." The probabilities of the event E_n in the two Markov chains are:

$$\mathbb{P}_{\mathcal{M}_1}(E_n) = \mathbb{P}_{\mathcal{M}_1}(X_n \le c_n) = \Phi(\tau\sqrt{n}) \qquad \mathbb{P}_{\mathcal{M}_1}(E_n) = \mathbb{P}_{\mathcal{M}_1}(Y_n \le c_n) = \Phi(\frac{-0.5\tau\sqrt{n}}{\sqrt{0.25-\tau^2}}),$$

where Φ is the CDF of the standard normal distribution. For $n \to \infty$ the probability of E_n in $\mathcal{M}_1$ and $\mathcal{M}_2$ converges to 1 and 0, respectively, so the total variation distance converges to 1.

4.4.2 Negative Result for the Total Variation Distance

Now we show that there is no statistical procedure for estimating total variation distance that would almost-surely terminate. From the second part of the proof, it also follows that there is no statistical algorithm even for fixed α and δ.

Theorem 4.1. *For any $\delta < 1$ and $\alpha < \frac{1}{2}$, there is no algorithm for computing a $1 - \alpha$ confidence interval of size δ for the total variation distance that almost-surely terminates.*

Proof. Let us write $\mathcal{M}(\tau)$ for a Markov chain in Figure 4.1 with the parameter τ and the initial distribution being stationary.

For $\alpha < \frac{1}{2}$ we define the following decision problem B_α:

- The input to B_α is a single path from $\mathcal{M}(\tau)$ of arbitrary length, where τ is unknown,
- The task of B_α is to output answer

  Yes with probability $\geq 1 - \alpha$ if $\mathsf{D}_{\mathsf{TV}}(\mathcal{M}(0), \mathcal{M}(\tau))) = 1$

  No with probability $\geq 1 - \alpha$ if $\mathsf{D}_{\mathsf{TV}}(\mathcal{M}(0), \mathcal{M}(\tau)) = 0$. Note that $\mathsf{D}_{\mathsf{TV}}(\mathcal{M}(0), \mathcal{M}(\tau))$ can equal only 0 or 1.

The remaining part of proof is done in two parts. In the first part, we show that there is no algorithm that solves B_α and almost-surely terminates. In the second part we reduce the problem B_α to computing a confidence interval for the total variation distance.

Part I. Suppose the opposite of the claim: that for some $\alpha < \frac{1}{2}$ there is an algorithm which solves B_α and almost-surely terminates. We represent the algorithm for solving B_α as a deterministic Turing machine TM, which works as follows:

1. The input tape of TM contains a (single) randomly sampled run of $\mathcal{M}(\tau)$,
2. TM reads a part of the run from the tape and eventually returns **Yes/No** answer.

The input to the TM is random, therefore we can assign a probability distribution to the computations of TM. To this end, we represent the answer of TM by the random variable $X : \mathsf{Runs} \mapsto \{\textbf{Yes}, \textbf{No}\}$, and we use the random variable $Y : \mathsf{Runs} \mapsto \mathbb{N} \cup \{\infty\}$ to represent

the number of path symbols TM reads before terminating, where ∞ means that TM does not terminate.

Suppose we run TM on the Markov chain $\mathcal{M}(0)$. We write $\mathbb{P}_1$ for the probability measure of TM on this input. The total variation distance between the two Markov chains $\mathcal{M}(0)$ is 0, so with probability $\geq 1-\alpha$ TM returns answer **No**, i.e. $\mathbb{P}_1(X = \textbf{No}) \geq 1-\alpha$.

By assumption TM almost-surely terminates on every input, so $\mathbb{P}_1(Y \in \mathbb{N}) = 1$. Let q be the following quantile:

$$q = \min\{c \in \mathbb{N} \ : \ \mathbb{P}_1(Y \leq c) \geq 0.5 + \alpha\}.$$

Claim. $q \in \mathbb{N}$ *(see below for the proof).*

It follows that:

$$\mathbb{P}_1(X = \textbf{No} \wedge Y \leq q) = 1 - \mathbb{P}_1(X = \textbf{Yes} \vee Y > q) \geq 1 - \mathbb{P}_1(X = \textbf{Yes}) - \mathbb{P}_1(Y > q) \geq 0.5. \tag{4.1}$$

The Turing machine TM is deterministic, so if it terminates after reading prefix π of some run ρ, then it terminates after reading prefix π of any run. As a consequence, the event $Y \leq q$ can be represented as a union of ℓ cones where $\ell \leq |2^{Ap}|^q = 2^q$ since $2^{Ap} = \{a, b\}$ in $\mathcal{M}$:

$$\{\rho \ : \ Y(\rho) \leq q\} = \bigcup_{i=1}^{\ell} \mathsf{Cone}(\pi_i),$$

where all $\pi_i \in (2^{Ap})^q$ are distinct. The event $X = \textbf{No} \wedge Y \leq q$ is a refinement of the event $Y \leq q$, so it may also be represented as

$$\{\rho \ : \ X = \textbf{No} \wedge Y(\rho) \leq q\} = \bigcup_{i=1}^{m} \mathsf{Cone}(\pi_i), \tag{4.2}$$

where $m \leq \ell \leq 2^q$. Since every path in $\mathcal{M}(0)$ of length q has probability 0.5^q, we get by (4.2)

$$\mathbb{P}_1(X = \textbf{No} \wedge Y(\rho) \leq q) = \mathbb{P}_1(\bigcup_{i=1}^{m} \mathsf{Cone}(\pi_i)) = \sum_{i=1}^{m} \mathbb{P}_1(\pi_i) = m0.5^q.$$

Then by (4.1) it follows that $m \geq 2^{q-1}$.

Now, we run TM on the Markov chain $\mathcal{M}(\epsilon)$ where $\epsilon = 0.5 - \alpha^{\frac{1}{q}} 2^{\frac{1-q}{q}}$ if $q > 0$ and $\epsilon = 0.25$ in the degenerated case of $q = 0$.

Claim. $\epsilon > 0$ *(see below for the proof).*

Let us write $\mathbb{P}_2$ for the probability measure of TM on the input $\mathcal{M}(\epsilon)$. The distance between $\mathcal{M}(0)$ and $\mathcal{M}(\epsilon)$ is 1, since $\epsilon > 0$. As a consequence, TM should return answer **Yes** on this input with probability $\geq 1 - \alpha$, or equivalently answer **No** with probability $< \alpha$. We show, however, that the probability of **No** is $\geq \alpha$:

$$\begin{aligned}
\mathbb{P}_2(X = \textbf{No} \wedge Y \leq q) &= \sum_{i=1}^{m} \mathbb{P}_2(\pi_i) && \text{by (4.2)} \\
&= \sum_{i=1}^{m} (0.5+\epsilon)^{u_i}(0.5-\epsilon)^{q-u_i} && u_i \text{ is number of } b\text{'s in } \pi_i \\
&\geq \sum_{i=1}^{m} (0.5-\epsilon)^{q} = m(0.5-\epsilon)^q \\
&\geq 2^{q-1}(0.5-\epsilon)^q = \alpha. && \text{by } m \geq 2^{q-1}..
\end{aligned}$$

We obtain a contradiction, thus the assumed machine TM does not exist.

Part II. Suppose for a contradiction that for some $\alpha < \frac{1}{2}, \delta < 1$ there exists an algorithm $\mathsf{Alg}_{\alpha,\delta}$ that solves the problem defined in the theorem and almost-surely terminates. Then then this algorithm can solve the problem B_α in the following way:

1. Use $\mathsf{Alg}_{\alpha,\delta}$ to compute a confidence interval I for the total variation distance between $\mathcal{M}(0)$ and $\mathcal{M}(\tau)$. Algorithm $\mathsf{Alg}_{\alpha,\delta}$ can sample any number of paths from $\mathcal{M}(0)$. Observe that in $\mathcal{M}(\tau)$ probability of seeing states a and b remains constant over time. Thus, sampling multiple paths from $\mathcal{M}(\tau)$ by $\mathsf{Alg}_{\alpha,\delta}$ can be replaced by sampling a single path from $\mathcal{M}(\tau)$.

2. Output **Yes** if $1 \in I$, **No** if $0 \in I$.

We have shown that for any $\alpha < \frac{1}{2}$ the problem B_α cannot by solved by an algorithm that almost-surely terminates. As a consequence, the algorithm $\mathsf{Alg}_{\alpha,\delta}$ cannot exist. □

Proofs for the claims We now prove claims that were used in the proof Theorem 4.1. First, we show that q as defined in the proof is finite.

Claim. $q \in \mathbb{N}$.

Proof. Suppose for contradiction that $q = \infty$, then

$$\forall c \in \mathbb{N} \ : \ \mathbb{P}_1(Y \leq c) < 0.5 + \alpha \tag{4.3}$$

From the standard results in probability theory we obtain

$$\lim_{c \to \infty^-} \mathbb{P}_1(Y \leq c) = \mathbb{P}_1(Y \in \mathbb{N}). \tag{4.4}$$

From the assumption that the algorithm terminates almost surely we get that the RHS of (4.4) equals 1, while the LHS must be $\leq 0.5 + \alpha < 1$ by (4.3), which is a contradiction. □

Second, we show that ϵ as defined in the proof of Theorem 4.1 is positive.

Claim. $\epsilon > 0$.

Proof. For $q = 0$ this is trivial. Otherwise, observe that the term $\alpha^{\frac{1}{q}} 2^{\frac{1-q}{q}}$ is monotonically increasing in α. Thus,

$$\alpha^{\frac{1}{q}} 2^{\frac{1-q}{q}} < 0.5^{\frac{1}{q}} 2^{\frac{1-q}{q}} = 0.5,$$

which implies that $\epsilon > 0$. □

4.5 Estimability: Finite-Trace Distance

In Section 4.5.1 we show how to estimate the distance given by traces of a fixed length. In Section 4.5.2 we show how to reduce the problem of computing the finite-trace distance D_{FT} (where traces of arbitrary lengths are considered) to computing a constant number of fixed-length distances.

4.5.1 Estimates for Fixed Length

Given two Markov chains $\mathcal{M}_1$ and $\mathcal{M}_2$ we wish to estimate the finite-trace distance for fixed length $k \in \mathbb{N}$

$$\mathsf{D}^k_{\mathsf{FT}} = \sup_{w \in 2^{Ap^k}} |\mathbb{P}_1(w) - \mathbb{P}_2(w)|.$$

There is $m = |2^{Ap}|^k$ words in $(2^{Ap})^k$ (we enumerate them as $w_1, \cdots, w_m$), so the traces of length k follow a multinomial distribution, i.e. for $i = 1, 2 \sum_{j=1}^{m}, \mathbb{P}_i(w_j) = 1$.

Condidence intervals We present a statistical procedure that estimates $\mathsf{D}^k_{\mathsf{FT}}$ with arbitrary precision. For $j \leq |2^{Ap}|^k$ we call a *contrast* Δ_j the difference in probabilities of trace w_j between $\mathcal{M}_1$ and $\mathcal{M}_2$: $\Delta_j = |\mathbb{P}_1(w_j) - \mathbb{P}_2(w_j)|$. The distance $\mathsf{D}^k_{\mathsf{FT}}$ is the maximum over all such contrasts $\mathsf{D}^k_{\mathsf{FT}} = \max_{j \leq m} \Delta_j$. We use the statistical procedure of [94] to simultaneously estimate all contrasts. We sample random paths from both Markov chains, and let n_i^j denote the number of observations of trace w_j in a Markov chain $\mathcal{M}_i$. We write $n_i = \sum_{j \leq m} n_i^j$ for the sum of all observations in $\mathcal{M}_i$. The estimator of $\mathbb{P}_i(w_j)$ is $\tilde{p}_i^j = \frac{n_i^j}{n_i}$, and the estimator of Δ_j is $\tilde{\Delta}_j = |\tilde{p}_1^j - \tilde{p}_2^j|$.

Theorem 4.2 ([94])**.** *As $n_1, n_2 \to \infty$ the probability approaches $1 - \alpha$ that simultaneously for all contrasts*

$$|\Delta_j - \tilde{\Delta}_j| \leq S_j M \qquad \textit{where} \qquad S_j = \sqrt{\frac{\tilde{p}_1^j - (\tilde{p}_1^j)^2}{n_1} + \frac{\tilde{p}_2^j - (\tilde{p}_2^j)^2}{n_2}},$$

and M is the square root of the $\frac{1-\alpha}{100}$ percentile of the χ^2 distribution with $|2^{Ap}|^k$ degrees of freedom.

The procedure for estimating $\mathsf{D}^k_{\mathsf{FT}}$ works as follows. For $\epsilon, \alpha > 0$ we sample paths from $\mathcal{M}_1$ and $\mathcal{M}_2$ until, by Theorem 4.2, with probability $1 - \alpha$ for all contrasts $|\Delta_j - \tilde{\Delta}_j| \leq \epsilon$. Then with probability $1 - \alpha$ it holds that $|\mathsf{D}^k_{\mathsf{FT}} - \max_{j \leq m} \tilde{\Delta}_j| \leq \epsilon$.

4.5.2 Estimates for Unbounded Length

Intuitively, the longer the path, the less probable it is, and the less distance it can cause. However, this is only true if along the path probabilistic choices are made repeatedly.

Definition 4.3 (Branching and deterministic state)**.** *In a Markov chain $\mathcal{M}$, a state $s \in S$ is* k-deterministic*, if there exists a word w of length k, such that $\mathbb{P}_s(w) = 1$. Otherwise, s is* k-branching*. A state $s \in S$ is* deterministic*, if it is k-deterministic for all $k \in \mathbb{N}$.*

Lemma 4.1. *If $s \in S$ is k-branching, it is also $(k+1)$-branching. Dually, if it is k-deterministic, it is also $(k-1)$-deterministic.*

Proof. The lemma follows trivially from the definition: if there exist two different words $w, w' \in (2^{Ap})^k$ such that $\mathbb{P}_s(w) > 0$ and $\mathbb{P}_s(w') > 0$, they can be always extended to different words $wa, w'a' \in (2^{Ap})^{k+1}$ with positive probability. □

Example 4.1. *Every state is trivially* 1*-deterministic. In Figure 4.4, the leftmost state is* 3*-deterministic and* 4*-branching. The states of the Markov chain in Figure 4.2 are deterministic.*

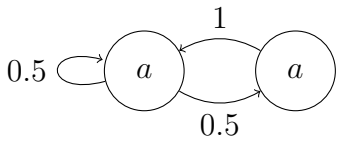

Figure 4.2: A Markov chains with deterministic states.

Lemma 4.2. *Consider a state* s *in a Markov chain* $\mathcal{M}$ *with* n *states. If state* s *is* n^2*-deterministic, then it is deterministic.*

Before proceeding to the proof, notice that even though it may seem that every branching state must be $n+1$ branching, this is not the case in general. Observe the counterexample in Fig. 4.3. The leftmost state is 6-deterministic (only the word $aaabaa$ can be generated), while $n = 4$.

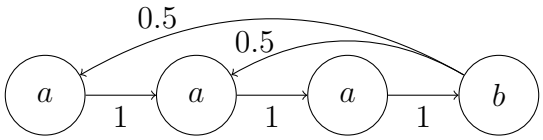

Figure 4.3: Markov chain with 4 states. The leftmost state is 6-deterministic, but not deterministic.

Proof. Consider state s that is n^2-deterministic and assume for contradiction that s is not deterministic. Let $N > n^2$ be the smallest number such that s is N-branching, and thus not $(N-1)$-branching. Then there exist two paths $\pi = s_1, s_2, \ldots, s_N$ and $\pi' = s_1, s'_2, \ldots, s'_N$ such that $s_1 = s$ and for $i = 1, 2, \ldots, N-1$, we have $L(s_i) = L(s'_i)$ and $L(s_N) \neq L(s'_N)$. Looking at a sequence of pairs $(s_1, s_1), (s_2, s'_2), \ldots, (s_{N-1}, s'_{N-1})$, since there are at most n^2 possible pairs of states over S, by the pigeon-hole principle at least two pairs will be repeating in the observed sequence, say $(s_i, s'_i) = (s_j, s'_j)$, where $i < j$. But then the paths $\pi'' = s_1, s_2, \ldots, s_i, s_{j+1}, \ldots, s_N$ and $\pi''' = s_1, s_2, \ldots, s_i, s_{j+1}, \ldots, s_N$ have $M < N$ states and they witness that s_1 is M-branching, which by Lemma 4.1 is in contradiction with s being $(N-1)$-deterministic. $\square$

Lemma 4.3. *If a state $s \in S$ is k-branching, then any word of length k starting from s has probability at most $(1 - p_{\mathsf{min}}^{k-1})$, i.e., $\forall w \in (2^{Ap})^k \ : \ \mathbb{P}_s(w) \leq 1 - p_{\mathsf{min}}^{k-1}$.*

To illustrate this, observe the Markov chain in Fig. 4.4 with leftmost initial state.

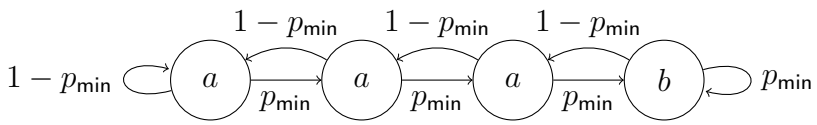

Figure 4.4: Markov chain with the leftmost initial state, s.t. $\mathbb{P}(a) = \mathbb{P}(aa) = \mathbb{P}(aaa) = 1$, $\mathbb{P}(aaab) = p_{\mathsf{min}}^3$, $\mathbb{P}(aaaa) = 1 - p_{\mathsf{min}}^3$.

Proof. Let $w \in (2^{Ap})^k$. Since s is k-branching, there exists a word $w' \in (2^{Ap})^k$ such that $w' \neq w$ and $\mathbb{P}_s(w') > 0$. Hence there exists at least one path with $k-1$ transitions, producing the trace w', and thus $\mathbb{P}_s(w') \geq p_{\mathsf{min}}^{k-1}$. Finally, $\mathbb{P}_s(w) \leq 1 - \mathbb{P}_s(w') \leq 1 - p_{\mathsf{min}}^{k-1}$. □

We show that, for estimating the finite trace distance with the required precision ϵ, it suffices to infer probabilities of the words up to some finite length k, which depends on ϵ. The idea is that paths that become deterministic before step k do not change their probability afterwards, while all other paths together have the probability bounded by ϵ.

Lemma 4.4. *Let s be a n^2-deterministic state in a Markov chain $\mathcal{M}$ with n states. Then there are words u, z, such that $|z| + |u| \leq n$, $|u| \geq 1$, and $\mathbb{P}_s(zu^\omega) = 1$.*

Proof. Consider any run $\rho = s_1 s_2 \cdots$, where $s_1 = s$. Let t be the first state on ρ that occurs twice, i.e. j is the smallest index such that

$$\exists i : \ i < j \ \wedge \ s_i = s_j = t.$$

Here s_i, s_j are the first and second occurrence of t on ρ, respectively. It holds that $j \leq n+1$, because otherwise some other state would occur twice earlier than t.

Let u, z be the following words $z = L(s_1 \cdots s_{i-1})$ and $u = L(s_i \ldots s_{j-1})$. Clearly $|z| + |u| \leq n$ and $|u| \geq 1$. The word u can be repeated any number of times from state $s_i = s_j = t$. By Lemma 4.2, state s is ∞-deterministic and thus t as well. Hence u^ω has the probability one from the state t. As a consequence $\mathbb{P}_s(zu^\omega) = 1$. □

We denote the k-prefix of $\overline{w}$ by $\overline{w} \downarrow k = A_0 \cdots A_k$ and similarly for prefixes of words. We write $\mathsf{pref}(\overline{w})$ for the set of all prefixes of $\overline{w}$:

$$\mathsf{pref}(\overline{w}) = \{w \mid \exists k \in \mathbb{N} \ : \ w = \overline{w} \downarrow k\}.$$

Definition 4.4. *A word $w \in (2^{Ap})^+$ is called* k-ultimately periodic in a Markov chain $\mathcal{M}$ *if* $\mathbb{P}(w) > 0$ *and there exists a word u such that $w \in \mathsf{pref}((2^{Ap})^k u^\omega)$ and $1 \leq |u| \leq n$, where n is the number of states in $\mathcal{M}$.* □

Intuitively, for sufficiently long word w and large ϵ, if $\mathbb{P}(w) > \epsilon$ and w is k-ultimately periodic, then it enters within k steps a BSCC, which is bisimilar to a cycle (all transition probabilities are 1). One can also prove that this is the only way for a ω-word to achieve a probability greater than ϵ.

For a word w we write $B^k(w)$ for the set of paths that are labelled by w, have a positive probability and where all states up to step k are n^2-branching:

$$B^k(w) = \{\pi = s_1 \cdots s_{|w|} \in L^{-1}(w) \mid \mathbb{P}(\pi) > 0 \wedge \forall i \leq \min(k, |w|) \ : \ s_i \text{ is } n^2\text{-branching}\}.$$

In a similar way, we write $D^k(w)$ for the set of paths that enter a (n^2-)deterministic state before step k

$$D^k(w) = \{\pi = s_1 \cdots s_{|w|} \in L^{-1}(w) \mid \mathbb{P}(\pi) > 0 \wedge \exists i \leq \min(k, |v|) \ : \ s_i \text{ is } n^2\text{-deterministic}\}.$$

For any k, we can partition paths labeled by w into B^k-paths and D^k-paths:

$$\mathbb{P}(w) = \sum_{\pi \in L^{-1}(w)} \mathbb{P}(\pi) = \sum_{\pi \in B^k(w)} \mathbb{P}(\pi) + \sum_{\pi \in D^k(w)} \mathbb{P}(\pi). \tag{4.5}$$

Now we show that the probability of B^k-paths diminishes exponentially with length k:

Lemma 4.5. *Consider a Markov chain $\mathcal{M}$ with n states. For every $k \in \mathbb{N}$ and word w, if $|w| > k$ then*

$$\sum_{\pi \in B^k(w)} \mathbb{P}(\pi) \leq (1 - p_{\mathsf{min}}^{n^2})^{\lfloor \frac{k}{n^2} \rfloor}.$$

Proof. Through the proof let $c = n^2$ and $w \uparrow n$ denote the suffix of w of length n. We show

$$\sum_{s_1 \cdots s_{|w|} \in B^k(w)} \mathbb{P}(s_1 \cdots s_{|w|})$$

$$\leq \sum_{s_1\cdots s_k \in B^k(w)} \mathbb{P}(s_1 \cdots s_k) \qquad s_1\cdots s_k = s_1\cdots s_{|w|}\downarrow k$$

$$= \sum_{\substack{s_1\cdots s_{k-c}\in \\ B^k(w\downarrow k-c)}} \mathbb{P}(s_1 \cdots s_{k-c}) \left(\sum_{\substack{s_{k-c}\cdots s_k \in \\ B^c(w\uparrow c+1)}} \mathbb{P}_{s_{k-c}}(s_{k-c} \cdots s_k) \right) \qquad \text{split } w \text{ into } w\downarrow k-c \text{ and } w\uparrow(c+1)$$

$$\leq \sum_{\substack{s_1\cdots s_{k-c}\in \\ B^k(w\downarrow(k-c))}} \mathbb{P}(s_1 \cdots s_{k-c}) \cdot \mathbb{P}_{s_{k-c}}(w \uparrow c+1) \qquad B^c(x)\subseteq L^{-1}(x)$$

$$\leq \sum_{\substack{s_1\cdots s_{k-c}\in \\ B^k(w\downarrow k-c)}} \mathbb{P}(s_1 \cdots s_{k-c})(1-p_{\mathsf{min}}^c) \qquad \text{by Lemma 4.3, since } s_{k-c} \text{ is } c\text{-branching, and thus } (c+1)\text{-branching}$$

$$\leq \sum_{\substack{s_1\cdots s_{k-2c}\in \\ B^k(w\downarrow k-2c)}} \mathbb{P}(s_1 \cdots s_{k-2c})(1-p_{\mathsf{min}}^c)^2 \qquad \text{by Lemma 4.3, since } s_{|w|-2c} \text{ is } c\text{-branching, and thus } (c+1)\text{-branching}$$

$$\leq \sum_{\substack{s_1\cdots s_{k-\lfloor \frac{k}{c}\rfloor c}\in \\ B^k(w\downarrow k-\lfloor \frac{k}{c}\rfloor c)}} \mathbb{P}(s_1 \cdots s_k)(1-p_{\mathsf{min}}^c)^{\lfloor \frac{k}{c}\rfloor} \qquad \text{by repeatedly applying Lemma 4.3}$$

$$\leq (1-p_{\mathsf{min}}^c)^{\lfloor \frac{k}{c}\rfloor}.$$

□

Lemma 4.6. *Let w be a word in a Markov chain $\mathcal{M}$ with n states. For every $\epsilon > 0$, if $\mathbb{P}(w) > \epsilon$ and $|w| > k$ then w is k-ultimately periodic in $\mathcal{M}$, where $k = n^2\lceil \frac{\log \epsilon}{\log(1-p_{\mathsf{min}}^{n^2})} \rceil + n$.*

Proof. Assume that $|w| > k$. We split paths labelled by w into $B^{k-n}(w)$ and $D^{k-n}(w)$ as in (4.5):

$$\mathbb{P}(w) = \sum_{s_1\cdots s_{|w|}\in L^{-1}(w)} \mathbb{P}(s_1 \cdots s_{|w|}) = \sum_{\substack{s_1\cdots s_{|w|}\in \\ B^{k-n}(w)}} \mathbb{P}(s_1 \cdots s_{|w|}) + \sum_{\substack{s_1\cdots s_{|w|}\in \\ D^{k-n}(w)}} \mathbb{P}(s_1 \cdots s_{|w|}). \tag{4.6}$$

By Lemma 4.5 we get

$$\sum_{s_1\cdots s_{|w|}\in B^{k-n}(w)} \mathbb{P}(s_1 \cdots s_{|w|}) \leq \epsilon. \tag{4.7}$$

Now, from the assumption $\mathbb{P}(w) > \epsilon$, (4.6) and (4.7), it follows that

$$\sum_{s_1\cdots s_{|w|}\in D^{k-n}(w)} \mathbb{P}(s_1 \cdots s_{|w|}) > 0.$$

This implies that there is a path $\pi = s_1 \cdots s_{|w|} \in D^{k-n}(w)$. By the definition of $D^{k-n}(w)$, π has a n^2-deterministic state before step $k-n$, and w.l.o.g. let s_{k-n} be that state. By Lemma 4.4,

every positive word from state s_{k-n} is a prefix of zu^ω for some words z, u such that $|z| + |u| \leq n$. Therefore $w \in \mathsf{pref}(yzu^\omega)$, where $y = L(s_1 \cdots s_{k-n})$, i.e. w is $|k|$-ultimately periodic. $\square$

Lemma 4.7. *Consider a Markov chain $\mathcal{M}$ with n states. Let w be a k-ultimately periodic word in $\mathcal{M}$, and x be a prefix of w such that $|x| > k + n$. Then*

$$\mathbb{P}(x) - \mathbb{P}(w) \leq (1 - p_{\mathsf{min}}^{n^2})^{\lfloor \frac{k-n}{n^2} \rfloor}.$$

Proof. Let $c = n^2$. We split $\mathbb{P}(x)$ and $\mathbb{P}(w)$ in the following way:

$$\mathbb{P}(x) = \overbrace{\sum_{s_1 \cdots s_{|x|} \in B^{k-n}(x)} \mathbb{P}(s_1 \cdots s_{|x|})}^{SB_1} + \overbrace{\sum_{s_1 \cdots s_{|x|} \in D^{k-n}(x)} \mathbb{P}(s_1 \cdots s_{|x|})}^{SD_1} \tag{4.8}$$

$$\mathbb{P}(w) = \underbrace{\sum_{s_1 \cdots s_{|w|} \in B^{k-n}(w)} \mathbb{P}(s_1 \cdots s_{|w|})}_{SB_2} + \underbrace{\sum_{s_1 \cdots s_{|w|} \in D^{k-n}(w)} \mathbb{P}(s_1 \cdots s_{|w|})}_{SD_2}. \tag{4.9}$$

By Lemma 4.5 we get

$$SB_1 \leq (1 - p_{\mathsf{min}}^{c})^{\lfloor \frac{k-n}{c} \rfloor} \tag{4.10}$$

$$SB_2 \leq (1 - p_{\mathsf{min}}^{c})^{\lfloor \frac{k-n}{c} \rfloor}. \tag{4.11}$$

We now prove that the deterministic paths for w and x have the same probability

$$SD_1 = SD_2. \tag{4.12}$$

Consider any path $\pi = s_1 \cdots s_{|x|} \in D^{k-n}(x)$. By definition path π enters a n^2-deterministic state before step $k - n$. W.l.o.g let s_{k-n} be that n^2-deterministic state. By Lemma 4.4 there are words z, u such that

$$\mathbb{P}_{s_{k-n}}(zu^\omega) = 1,$$

and $|u| \geq 1$, $|z| + |u| \leq n$. Thus the word labelling π has the form

$$x = L(\pi) \in \mathsf{pref}(yzu^\omega),$$

where $y = L(s_1 \cdots s_{k-n})$. Both x and w are k-ultimately periodic and have length greater than $k + n$, so they both must be of the form

$$w, x = L(\pi) \in \mathsf{pref}(yzu^\omega).$$

Consider any path $\pi \in D^{k-n}(x)$, and let $E(\pi)$ denote all extensions of π to the paths of length $|w|$:

$$E(\pi) = \{p_1 \cdots p_{|w|} \mid \forall i \leq |x| \ : \ p_i = s_i \wedge \forall i < |w| \ : \ \mathbf{P}(p_i, p_{i+1}) > 0\}.$$

All paths in $E(\pi)$ are labelled by the same word, namely w, and enter a n^2-deterministic state before step $k-n$, therefore $E(\pi) \subseteq D^{k-n}(w)$, which implies that $\mathbb{P}(SB_1) \leq \mathbb{P}(SB_2)$. Now, consider any path $\pi \in D^{k-n}(w)$. The prefix of $\pi \downarrow |x|$ is labelled by the word x, and enters a n^2-deterministic state before step $k-n \geq |x|$, so $\pi \downarrow |x| \in D^{k-n}(w)$; this implies the other inequality that $\mathbb{P}(SB_2) \leq \mathbb{P}(SB_1)$.

Finally, we write

$$\begin{aligned}
&\mathbb{P}(x) - \mathbb{P}(w) = SB_1 + SD_1 - SB_2 - SD_2 && \text{by (4.8) and (4.9)}\\
&= SB_1 - SB_2 && \text{by (4.12)}\\
&\leq (1 - p_{\mathsf{min}}^c)^{\lfloor \frac{k-n}{c} \rfloor} && \text{by (4.10) and (4.11).}
\end{aligned}$$

□

Theorem 4.3. *Consider Markov chains $\mathcal{M}_1$ and $\mathcal{M}_2$ that have at most n states. For $\epsilon > 0$ it holds that*

$$|\mathsf{D}_{\mathsf{FT}}(\mathcal{M}_1, \mathcal{M}_2) - \max_{i \leq k} \mathsf{D}^i_{\mathsf{FT}}(\mathcal{M}_1, \mathcal{M}_2)| \leq \epsilon, \quad \textit{where} \quad k = n^2 \lceil \frac{\log \epsilon}{\log(1 - p_{\mathsf{min}}^{n^2})} \rceil + 2n.$$

Proof. We show that for any word $w \in (2^{Ap})^+$:

$$\Big| |\mathbb{P}_1(w) - \mathbb{P}_2(w)| - |\mathbb{P}_1(w \downarrow k) - \mathbb{P}_2(w \downarrow k)| \Big| \leq \epsilon. \tag{4.13}$$

For $|w| \leq k$ (4.13) holds trivially. Suppose that $|w| \geq k$ and consider two cases.

1. If $\mathbb{P}_i(w \downarrow k) > \epsilon$, then by Lemma 4.6 $w \downarrow k$ is $(k-n)$-ultimately periodic. Then by Lemma 4.7 $\mathbb{P}_i(w \downarrow k) \leq \mathbb{P}_i(w) + \epsilon$.

2. If $\mathbb{P}_i(w \downarrow k) \leq \epsilon$, then clearly $\mathbb{P}_i(w \downarrow k) \leq \mathbb{P}_i(w) + \epsilon$.

Both cases can be summarised by

$$\mathbb{P}_i(w) \leq \mathbb{P}_i(w \downarrow k) \leq \mathbb{P}_i(w) + \epsilon. \tag{4.14}$$

W.l.o.g assume that $\mathbb{P}_1(w) \geq \mathbb{P}_2(w)$. Then by (4.14)

$$\mathbb{P}_1(w \downarrow k) - \mathbb{P}_2(w \downarrow k) \geq \mathbb{P}_1(w) - \mathbb{P}_2(w) - \epsilon,$$

which implies (4.13). □

4.6 Other Distances

We now discuss the consequences of the (in)estimability results for several specific subclasses of ω-regular languages, captured topologically, logically, or by automata. We also remark on the estimability in case when the transition probabilities have finite precision.

4.6.1 Topology

Negative result for clopen sets Note that the proof of inestimability was based on the ability to express the events E_n for any $n \in \mathbb{N}$:

$$E_n = \text{“there is at most } c_n = (0.5 + \tau/2)n \text{ symbols } b \text{ in the prefix path of length } n\text{.”}$$

Observe that each E_n can be expressed as a finite union of cones, each expressing exact positions of a's and b's in the first n steps. For instance, for $\tau = 0.2$, the event E_2, “there is at most 1 symbol b in the first 2 steps,” can be described by the union $\mathsf{Cone}(aa) \cup \mathsf{Cone}(ab) \cup \mathsf{Cone}(ba)$.

Since finite unions of cones form exactly the clopen sets, the lowest class Δ_1 in the Borel hierarchy, it follows that distances based on any class in the hierarchy are inestimable.

Positive result for the infinite-trace distance Using the result on finite-trace distance, we can prove that the infinite-trace distance D_{IT} of Example 4.3.1 is also estimable. Indeed, the distance is non-zero only due to k-ultimately periodic ω-words with positive probability. By Lemma 4.7 we can provide confidence intervals for these probabilities through the k-prefixes using the fixed-length distance $\mathsf{D}^k_{\mathsf{FT}}$.

4.6.2 Logic

Negative result for LTL**.** The LTL distance as in Example 4.3.1 is again inestimable since we can express the event E_n in LTL by a finite composition of operators $\bigcirc, \wedge, \vee$ (notably

this fragment induces the same distance as LTL [86]). Indeed, for instance, for $\tau = 0.2$, the event E_{10}, "there is at most 6 symbols b in the path prefix of length 10," is equivalent to "at least 4 symbols a in the path prefix of length n," and it can be described by a disjunction of $\binom{10}{4}$ formulae, each determining the possible position of symbols a, resulting in a formula $(a \wedge \bigcirc a \wedge \bigcirc^2 a \wedge \bigcirc^3 a) \vee (a \wedge \bigcirc a \wedge \bigcirc^2 a \wedge \bigcirc^4 a) \vee \ldots \vee (\bigcirc^7 a \wedge \bigcirc^8 a \wedge \bigcirc^9 a \wedge \bigcirc^{10} a)$.

Positive result for $\mathrm{LTL}(\Diamond\Box, \Box\Diamond)$**.** The distance generated by the fragment of LTL described by combining operators $\Diamond\Box$ and $\Box\Diamond$ and Boolean operators is estimable. Notice that the probability of the property $\varphi \equiv \Diamond\Box\varphi'$ equals the probability of reaching a BSCC such that φ' holds in all of its states, while the probability of property $\varphi \equiv \Box\Diamond\varphi'$ equals the probability that every BSCC contains a state which satisfies φ'. Hence, properties expressed in this fragment of LTL can be checked by inferring all BSCCs of a chain and a simple analysis of them. The statistical estimation of all BSCCs for labelled Markov chains where only the minimal transition probability is known is possible and is shown in Chapter 3.

4.6.3 Automata

Negative result for automata distances. For the class of all deterministic Rabin automata (DRA), the distance (as in Example 4.3.1) is inestimable. This is implied by the inestimability for clopen sets or for LTL. Further, we can also directly encode the event E_n that "at least k symbols a are observed in the path of length n" by an automaton: the DRA counts how many symbols a are seen in the prefix up to length n; this can be done with $k \cdot n$ states where the automaton is in a state $s_{k',n'}$ if and only if in the $n' \leq n$ prefix of the input word, there are $k' \leq k$ symbols a.

Positive result for fixed-size automata. When restricting to the class of DRA of size at most $k \in \mathbb{N}$, the distance $\mathsf{D}_{DRA \leq k}$ can be estimated. A naive algorithm amounts to enumerating all automata up to given size k, then applying statistical model checking to infer the probability of satisfying the automata in each of the Markov chains, and checking for which automaton the probability difference in the two chains is maximized. Statistically inferring the probability of whether a (black-box) Markov chain satisfies a property given by a DRA is a subroutine of the procedure for statistical model checking Markov chains for LTL, described in Chapter 3.

4.6.4 Finite Precision

When the transition probabilities have finite precision, e.g. are given by at most two decimal digits, several negative results turn positive. Finite precision allows us to learn the Markov chains exactly with high probability, by rounding the learnt transition probabilities to the closest multiple of the precision. Subsequently, we can approximate the distance by the algorithms applicable in the white-box setting. In case of the total variation distance, one can apply the approximation algorithm of [83]; for trace distances, the approximation algorithm of [86] is also available. In particular, for the special case of the trace equivalence E_{FT} we can leverage the fact that Markov chains are equivalent when all their traces up to length $|\mathcal{M}_1| + |\mathcal{M}_2| - 1$ have equal probability. With the assumption of finite precision one can get by sampling the exact distribution of such traces with high confidence. Note that the same algorithm can not be applied without assuming finite precision, since arbitrarily small difference in chains cannot be detected.

5 Qualitative Analysis of Probabilistic Systems

5.1 Introduction

One of the key challenges in analysis of probabilistic systems (as in the case of non-probabilistic systems) is the *state explosion* problem [9], as the size of concurrent systems grows exponentially in the number of components. One key technique to combat the state explosion problem is the *assume-guarantee* style composition reasoning [95], where the analysis problem is decomposed into components and the results for components are used to reason about the whole system, instead of verifying the whole system directly. For a system with two components, the compositional reasoning can be captured as the following simple rule: consider a system with two components G_1 and G_2, and a specification G' to be satisfied by the system; if A is an abstraction of G_2 (i.e., G_2 refines A) and G_1 in composition with A satisfies G', then the composite systems of G_1 and G_2 also satisfies G'. Intuitively, A is an assumption on G_1's environment that can be ensured by G_2. This simple, yet elegant asymmetric rule is very effective in practice, specially with a *counterexample guided abstraction-refinement* (CEGAR) loop [35]. There are many symmetric [96] as well as circular compositional reasoning [97; 96; 98] rules; however the simple asymmetric rule is most effective in practice and extensively studied, mostly for non-probabilistic systems [96; 99; 100; 101].

In this chapter we consider the fragment of pCTL* [50; 51; 52] that is relevant for *qualitative analysis*, and refer to this fragment as QCTL^*. The qualitative analysis for probabilistic systems refers to *almost-sure* (resp. *positive*) properties that are satisfied with probability 1 (resp. positive probability). The qualitative analysis for probabilistic systems is an important problem in

verification that is of interest independently of the quantitative analysis problem. There are many applications where we need to know whether the correct behavior arises with probability 1. For instance, when analyzing a randomized embedded scheduler, we are interested in whether every thread progresses with probability 1 [102]. Even in settings where it suffices to satisfy certain specifications with probability $\lambda < 1$, the correct choice of λ is a challenging problem, due to the simplifications introduced during modeling. For example, in the analysis of randomized distributed algorithms it is quite common to require correctness with probability 1 (see, e.g., [103; 104]). Furthermore, in contrast to quantitative analysis, qualitative analysis is robust to numerical perturbations and modeling errors in the transition probabilities.

Contributions In this chapter we focus on the compositional reasoning of probabilistic systems with respect to qualitative properties, and our main contribution is a CEGAR approach for qualitative analysis of probabilistic systems. The details of our contributions are as follows:

1. To establish the logical relation induced by QCTL^* we consider the logic ATL^* for two-player games and the two-player game interpretation of an MDP where the probabilistic choices are resolved by an adversary. In case of non-probabilistic systems and games there are two classical notions for refinement, namely, *simulation* [19] and *alternating simulation* [21]. We first show that the logical relation induced by QCTL^* is *finer* than the intersection of simulation and alternating simulation. We then introduce a new notion of simulation, namely, *combined simulation*, and show that it captures the logical relation induced by QCTL^*.

2. We show that our new notion of simulation, which captures the logic relation of QCTL^*, can be computed using discrete graph algorithms in quadratic time. In contrast, the current best known algorithm for strong simulation is polynomial of degree seven and requires numerical algorithms. The other advantage of our approach is that it can be applied uniformly both to qualitative analysis of probabilistic systems as well as analysis of two-player games (that are standard models for open non-probabilistic systems).

3. We present a CEGAR approach for the computation of combined simulation, and the counterexample analysis and abstraction refinement is achieved using the ideas of [105] proposed for abstraction-refinement for games.

4. We have implemented our approach both for qualitative analysis of MDPs as well as games, and experimented on a number of well-known examples of MDPs and games. Our experimental results show that our method achieves significantly better performance as compared to the non-compositional verification as well as compositional analysis of MDPs with strong simulation.

Outline The basic definitions of games, logic and Markov decision processes (MDP) are in Chapter 2. In Section 5.2 we review the related work. In Section 5.3 we introduce additional notions for games. In Section 5.4 we present a new simulation relation for games, show that it is finer than both simulation and alternating simulation, and present algorithms to compute the relation. In Section 5.5 we present a qualitative logics for MDPs, and in Section 5.6 show that the logical relation induced by the qualitative logics on MDPs can be obtained through our simulation relation introduced in Section 5.4. In Section 5.7 we present a CEGAR approach for our simulation relation and present experimental results in Section 5.8.

5.2 Related Work

There are many works that have studied abstraction-refinement and compositional analysis for probabilistic systems [106; 107; 108; 109]. Our work is most closely related to and inspired by [110] where a CEGAR approach was presented for analysis of MDPs (or labeled probabilistic transition systems); and the refinement relation was captured by *strong simulation* that captures the logical relation induced by safe-pCTL [50; 51; 52].

Compositional and assume-guarantee style reasoning has been extensively studied mostly in the context of non-probabilistic systems [96; 99; 100; 101]. Game-based abstraction refinement has been studied in the context of probabilistic systems [108]. The CEGAR approach has been adapted to probabilistic systems for reachability [107] and safe-pCTL [106] under monolithic (non-compositional) abstraction refinement. The work of [110] considers CEGAR for compositional analysis of probabilistic system with strong simulation. The main difference w.r.t. [110] is that strong simulation preserves exact probabilities and therefore the algorithm of [110] requires numerical algorithms whereas our algorithm requires only discrete graph algorithms. Moreover, our approach can be applied uniformly both to MDPs and two-player games.

An abstraction-refinement algorithm for a class of quantitative properties was studied in [111; 112] and also implemented [113]. Our logical characterization of the simulation relation is similar in spirit to [114], which shows how a fragment of the modal μ-calculus can be used to efficiently decide behavioral preorders between components.

The qualitative analysis problem has been extensively studied for many probabilistic models, such as for MDPs [115; 15; 116; 117; 118; 119; 120], perfect-information stochastic games [121; 122; 123; 124; 125], concurrent stochastic games [126; 127; 128; 129; 130; 131; 132], partial-observation MDPs [133; 134; 135; 136; 137], partial-observation stochastic games [138; 139; 140; 141; 142; 143; 144], and real-timed systems [145; 146].

Our work focuses on CEGAR for compositional analysis of probabilistic systems for qualitative analysis: we characterize the required simulation relation; present a CEGAR approach for the computation of the simulation relation; and show the effectiveness of our approach both for qualitative analysis of MDPs and games.

5.3 Games and Logic

In Chapter 2 we presented two-player games and alternating-time temporal logic (ATL^*) for specifying their properties. In this section we present additional notions that are used later in this chapter. For technical convenience we consider that in the case of turn-based games, there is an atomic proposition $\mathsf{turn} \in Ap$ such that for every Player-1 state s we have $\mathsf{turn} \in L(s)$, and for every Player 2 state s' we have $\mathsf{turn} \notin L(s')$.

Definition 5.1 (Parallel composition of two-player games)**.** *Given games* $G = (S, A, \mathsf{Av}, \delta, L, s_0)$ *and* $G' = (S', A, \mathsf{Av}', \delta', L', s'_0)$ *the* parallel composition *of the games* $G \parallel G' = (\overline{S}, A, \overline{\mathsf{Av}}, \overline{\delta}, \overline{L}, \overline{s}_0)$ *is defined as follows:*

- *The states of the composition are* $\overline{S} = S \times S'$.
- *The set of actions does not change with the composition.*
- *For all* (s, s') *we have* $\overline{\mathsf{Av}}((s, s')) = \mathsf{Av}(s) \cap \mathsf{Av}'(s')$.
- *The transition function for a state* $(s, s') \in \overline{S}$ *and an action* $a \in \overline{\mathsf{Av}}((s, s'))$ *is defined as* $\overline{\delta}((s, s'), a) = \{(t, t') \mid t \in \delta(s, a) \wedge t' \in \delta'(s', a)\}$.

- *The labeling function* $\overline{L}((s, s'))$ *is defined as* $L(s) \cup L'(s')$.

- *The initial state is* $\overline{s}_0 = (s_0, s'_0)$.

For simplicity we assume that the set of actions in both components is identical, and for every pair of states the intersection of their available actions is non-empty. Parallel composition can be extended to cases where the sets of actions are different [147].

Logic fragments We define several fragments of the logic ATL^*:

- *Restricted temporal operator use.* An important fragment of ATL^* is ATL where every temporal operator is immediately preceded by a path quantifier.

- *Restricting path quantifiers.* We also consider fragments of ATL^* (resp. ATL) where the path quantifiers are restricted. We consider (i) 1-fragment (denoted $1\text{-}\mathrm{ATL}^*$) where only $\langle\langle 1 \rangle\rangle$ path quantifier is used; (ii) the $(1, 2)$-fragment (denoted $(1, 2)\text{-}\mathrm{ATL}^*$) where only $\langle\langle 1, 2 \rangle\rangle$ path quantifier is used; and (iii) the combined fragment (denoted $\mathrm{C}\text{-}\mathrm{ATL}^*$) where both $\langle\langle 1 \rangle\rangle$ and $\langle\langle 1, 2 \rangle\rangle$ path quantifiers are used. We use a similar notation for the respective fragments of ATL formulas.

Logical characterization of states Given two games G and G', and a logic fragment $\mathcal{F}$ of ATL^*, we consider the following relations on the state space induced by the logic fragment $\mathcal{F}$:

$$\preccurlyeq_{\mathcal{F}} (G, G') = \{(s, s') \in S \times S' \mid \forall \psi \in \mathcal{F} : \text{ if } s \models \psi \text{ then } s' \models \psi\};$$

and when the games are clear from context we simply write $\preccurlyeq_{\mathcal{F}}$ for $\preccurlyeq_{\mathcal{F}} (G, G')$. We will use the following notations for the relation induced by the logic fragments we consider: (i) $\preccurlyeq_1^*$ (resp. $\preccurlyeq_1$) for the relation induced by the $1\text{-}\mathrm{ATL}^*$ (resp. $1\text{-}\mathrm{ATL}$) fragment; (ii) $\preccurlyeq_{1,2}^*$ (resp. $\preccurlyeq_{1,2}$) for the relation induced by the $(1, 2)\text{-}\mathrm{ATL}^*$ (resp. $(1, 2)\text{-}\mathrm{ATL}$) fragment; and (iii) $\preccurlyeq_C^*$ (resp. $\preccurlyeq_C$) for the relation induced by the $\mathrm{C}\text{-}\mathrm{ATL}^*$ (resp. $\mathrm{C}\text{-}\mathrm{ATL}$) fragment. Given G and G' we can also consider G'' which is the disjoint union of the two games, and consider the relations on G''; and hence we will often consider a single game as input for the relations.

5.4 Combined Simulation

In this section we first recall the notion of simulation [19] and alternating simulation [21]; and then present a new notion of *combined simulation*.

Definition 5.2 (Simulation)**.** *Given two-player games* $G = (S, A, \mathsf{Av}, \delta, L, s_0)$ *and* $G' = (S', A', \mathsf{Av}', \delta', L', s'_0)$*, a relation* $\mathcal{S} \subseteq S \times S'$ *is a* simulation *from* G *to* G' *if for all* $(s, s') \in \mathcal{S}$ *the following conditions hold:*

1. Proposition match: *The atomic propositions match, i.e.,* $L(s) = L'(s')$.

2. Step-wise simulation condition: *For all actions* $a \in \mathsf{Av}(s)$ *and states* $t \in \delta(s, a)$ *there exists an action* $a' \in \mathsf{Av}'(s')$ *and a state* $t' \in \delta'(s', a')$ *such that* $(t, t') \in \mathcal{S}$.

We denote by $\mathcal{S}^{G,G'}_{\max}$ the largest simulation relation between the two games (we write $\mathcal{S}_{\max}$ instead of $\mathcal{S}^{G,G'}_{\max}$ when G and G' are clear from the context). We write $G \leqslant_{\mathcal{S}} G'$ when $(s_0, s'_0) \in \mathcal{S}_{\max}$. The largest simulation relation characterizes the logic relation of $(1, 2)$-ATL and $(1, 2)$-ATL*: the $(1, 2)$-ATL-fragment interprets a game as a transition system and the formulas coincide with existential CTL, and hence the logic characterization follows from the classical results on simulation and CTL [19; 147].

Proposition 5.1. *For all games* G *and* G' *we have* $\mathcal{S}^{G,G'}_{\max} = \preccurlyeq^*_{1,2} = \preccurlyeq_{1,2}$.

Definition 5.3 (Alternating simulation)**.** *Given two games* $G = (S, A, \mathsf{Av}, \delta, L, s_0)$ *and* $G' = (S', A', \mathsf{Av}', \delta', L', s'_0)$*, a relation* $\mathcal{A} \subseteq S \times S'$ *is an* alternating simulation *from* G *to* G' *if for all* $(s, s') \in \mathcal{A}$ *the following conditions hold:*

1. Proposition match: *The atomic propositions match, i.e.,* $L(s) = L'(s')$.

2. Step-wise alternating-simulation condition: *For all actions* $a \in \mathsf{Av}(s)$ *there exists an action* $a' \in \mathsf{Av}'(s')$ *such that for all states* $t' \in \delta'(s', a')$ *there exists a state* $t \in \delta(s, a)$ *such that* $(t, t') \in \mathcal{A}$.

We denote by $\mathcal{A}^{G,G'}_{\max}$ the largest alternating-simulation relation between the two games (we write $\mathcal{A}_{\max}$ instead of $\mathcal{A}^{G,G'}_{\max}$ when G and G' are clear from the context). We write $G \leqslant_{\mathcal{A}} G'$ when $(s_0, s'_0) \in \mathcal{A}_{\max}$. The largest alternating-simulation relation characterizes the logic relation of 1-ATL and 1-ATL* [21].

Proposition 5.2. *For all games G and G' we have $\mathcal{A}^{G,G'}_{\max} = \preccurlyeq^*_1 = \preccurlyeq_1$.*

We present a new notion of combined simulation that extends both simulation and alternating simulation, and we show how the combined simulation characterizes the logic relation induced by C-ATL* and C-ATL. Intuitively, the requirements on the combined-simulation relation combine the requirements imposed by alternating simulation and simulation in a step-wise fashion.

Definition 5.4 (Combined simulation)**.** *Given two-player games $G = (S, A, \mathsf{Av}, \delta, L, s_0)$ and $G' = (S', A', \mathsf{Av}', \delta', L', s'_0)$, a relation $\mathcal{C} \subseteq S \times S$ is a* combined simulation *from G to G' if for all $(s, s') \in \mathcal{C}$ the following conditions hold:*

1. Proposition match: *The atomic propositions match, i.e., $L(s) = L'(s')$.*
2. Step-wise simulation condition: *For all actions $a \in \mathsf{Av}(s)$ and states $t \in \delta(s, a)$ there exists an action $a' \in \mathsf{Av}'(s')$ and a state $t' \in \delta(s', a')$ such that $(t, t') \in \mathcal{C}$.*
3. Step-wise alternating-simulation condition: *For all actions $a \in \mathsf{Av}(s)$ there exists an action $a' \in \mathsf{Av}'(s')$ such that for all states $t' \in \delta'(s', a')$ there exists a state $t \in \delta(s, a)$ such that $(t, t') \in \mathcal{C}$.*

We denote by $\mathcal{C}^{G,G'}_{\max}$ the largest combined-simulation relation between the two games (and write $\mathcal{C}_{\max}$ when G and G' are clear from the context). We also write $G \leqslant_{\mathcal{C}} G'$ when $(s_0, s'_0) \in \mathcal{C}_{\max}$. We first illustrate with an example that the logic relation $\preccurlyeq_C$ induced by C-ATL is finer than the intersection of simulation and alternating-simulation relation; then present a game theoretic characterization of $\mathcal{C}_{\max}$; and finally show that $\mathcal{C}_{\max}$ gives the relations $\preccurlyeq^*_C$ and $\preccurlyeq_C$.

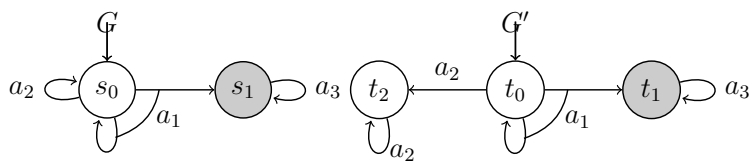

Figure 5.1: Games G, G' such that $G \leqslant_{\mathcal{S}} G'$ and $G \leqslant_{\mathcal{A}} G'$, but $G \not\leqslant_{\mathcal{C}} G'$.

Example 5.1. *Consider the games G and G' shown in Figure 5.1. White nodes are labeled by an atomic proposition p and gray nodes by q. The largest simulation and alternating-simulation relations between G and G' are: $\mathcal{S}^{G,G'}_{\max} = \{(s_0, t_0), (s_1, t_1)\}$, $\mathcal{A}^{G,G'}_{\max} = \{(s_0, t_0), (s_0, t_2), (s_1, t_1)\}$. However, consider the formula $\psi = \langle\langle 1 \rangle\rangle(\bigcirc(p \wedge \langle\langle 1, 2 \rangle\rangle(\bigcirc q)))$. We have that $s_0 \models \psi$, but $t_0 \not\models \psi$. It follows that $(s_0, t_0) \notin \preccurlyeq_C$.*

The simulation and the alternating-simulation relation can be obtained by solving two-player safety games [148; 21; 149]. We now define a two-player game for the combined-simulation relation characterization. The game is played on the synchronized product of the two input games. Given a state (s, s'), first Player 2 decides whether to check for the step-wise simulation condition or the step-wise alternating-simulation condition. The step-wise simulation condition is checked by playing a two-step game, and the step-wise alternating-simulation condition is checked by playing a four-step game.

Definition 5.5 (Combined-simulation games)**.** *Consider two games* $G = (S, A, \mathsf{Av}, \delta, L, s_0)$ *and* $G' = (S', A', \mathsf{Av}', \delta', L', s'_0)$*. We construct the* combined-simulation game

$$G^{\mathcal{C}} = (S^{\mathcal{C}}, A^{\mathcal{C}}, \mathsf{Av}^{\mathcal{C}}, \delta^{\mathcal{C}}, L^{\mathcal{C}}, s_0^{\mathcal{C}})$$

as follows:

- The set of states. *The set of states* $S^{\mathcal{C}}$ *is:*

$$\begin{aligned} S^{\mathcal{C}} = \quad & (S \times S') \cup (S \times S' \times \{\mathsf{Sim}\} \times \{1, 2\}) \cup (S \times S' \times \{\mathsf{Alt}\} \times \{2\}) \\ \cup \quad & (S \times S' \times \{\mathsf{Alt}\} \times A \times \{1\}) \cup (S \times S' \times \{\mathsf{Alt}\} \times A \times A' \times \{1, 2\}) \end{aligned}$$

  Intuitively, in states in $S \times S'$ *and in states where the last component is 2 it is Player 2's turn to make the choice of successors, and in all other states Player 1 makes the choice of actions.*

- The set of actions *The set of actions is as follows:* $A^{\mathcal{C}} = \{\bot\} \cup S \cup S' \cup A'$.

- The transition function and the action-available function

  1. Choice of simulation or alternating-simulation *For a state* (s, s') *we have only one action* $\bot$ *available for Player 1 and we have* $\delta^{\mathcal{C}}((s, s'), \bot) = \{(s, s', \mathsf{Alt}, 2), (s, s', \mathsf{Sim}, 2)\}$*, i.e., Player 2 decides whether to check for step-wise simulation or step-wise alternating-simulation conditions.*

  2. Checking step-wise simulation conditions *We describe the transitions for checking the simulation conditions:*

     (a) *For a state* $(s, s', \mathsf{Sim}, 2)$ *we have only one action* $\bot$ *available for Player 1 and we have* $\delta^{\mathcal{C}}((s, s', \mathsf{Sim}, 2), \bot) = \{(t, s', \mathsf{Sim}, 1) \mid \exists a \in \mathsf{Av}(s) : \ t \in \delta(s, a)\}$.

(b) *For a state* $\overline{s} = (t, s', \mathsf{Sim}, 1)$ *we have* $\mathsf{Av}^{\mathcal{C}}(\overline{s}) = \{t' \mid \exists a' \in \mathsf{Av}(s') : t' \in \delta'(s', a')\}$ *and* $\delta^{\mathcal{C}}(\overline{s}, t') = \{(t, t')\}$.

Intuitively, first Player 2 chooses an action $a \in \mathsf{Av}(s)$ *and a successor* $t \in \delta(s, a)$ *and challenges Player 1 to match, and Player 1 responds with an action* $a' \in \mathsf{Av}'(s')$ *and a state* $t' \in \delta'(s', a')$.

3. Checking step-wise alternating-simulation conditions *We describe the transitions for checking the alternating-simulation conditions:*

   (a) *For a state* $(s, s', \mathsf{Alt}, 2)$ *we have only one action* $\perp$ *available for Player 1 and we have* $\delta^{\mathcal{C}}((s, s', \mathsf{Alt}, 2), \perp) = \{(s, s', \mathsf{Alt}, a, 1) \mid a \in \mathsf{Av}(s)\}$.

   (b) *For a state* $\overline{s} = (s, s', \mathsf{Alt}, a, 1)$ *we have* $\mathsf{Av}^{\mathcal{C}}(\overline{s}) = \mathsf{Av}'(s')$ *and* $\delta^{\mathcal{C}}(\overline{s}, a') = \{(s, s', \mathsf{Alt}, a, a', 2)\}$.

   (c) *For a state* $(s, s', \mathsf{Alt}, a, a', 2)$ *we have only one action* $\perp$ *available for Player 1 and we have* $\delta^{\mathcal{C}}((s, s', \mathsf{Alt}, a, a', 2), \perp) = \{(s, t', \mathsf{Alt}, a, a', 1) \mid t' \in \delta'(s', a')\}$.

   (d) *For a state* $\overline{s} = (s, t', \mathsf{Alt}, a, a', 1)$ *we have* $\mathsf{Av}^{\mathcal{C}}(\overline{s}) = \delta(s, a)$ *and* $\delta^{\mathcal{C}}(\overline{s}, t) = \{(t, t')\}$.

   Intuitively, first Player 2 chooses an action a *from* $\mathsf{Av}(s)$ *and Player 1 responds with an action* $a' \in \mathsf{Av}'(s')$ *(in the first two-steps); then Player 2 chooses a successor* t' *from* $\delta'(s', a')$ *and Player 1 responds by choosing a successor* t *in* $\delta(s, a)$.

- The labeling function *The set of atomic proposition* Ap *contains a single proposition* $p \in Ap$. *The labeling function* $L^{\mathcal{C}}$ *given a state* $\overline{s} \in S^{\mathcal{C}}$ *is defined as follows:* $L^{\mathcal{C}}(\overline{s}) = p$ *iff* $\overline{s} = (s, s')$ *and* $L(s) \neq L'(s')$. *Intuitively, Player 2's goal is to reach a state* (s, s') *where the propositional labeling of the original games do not match, i.e., to reach a state labeled* p *by* $L^{\mathcal{C}}$.

- The initial state *The state* $s_0^{\mathcal{C}}$ *is* (s_0, s_0').

In the combined simulation game we refer to Player 1 as the *proponent* (trying to establish the combined simulation) and Player 2 as the *adversary* (trying to violate the combined simulation).

Example 5.2. *A part of the combined-simulation game of* G *and* G' *from Figure 5.1 is shown in Figure 5.2. Dashed arrows indicate that the successors of a given state are omitted in the figure. Gray states are labeled by an atomic proposition* p, *hence are the goal states for the adversary.*

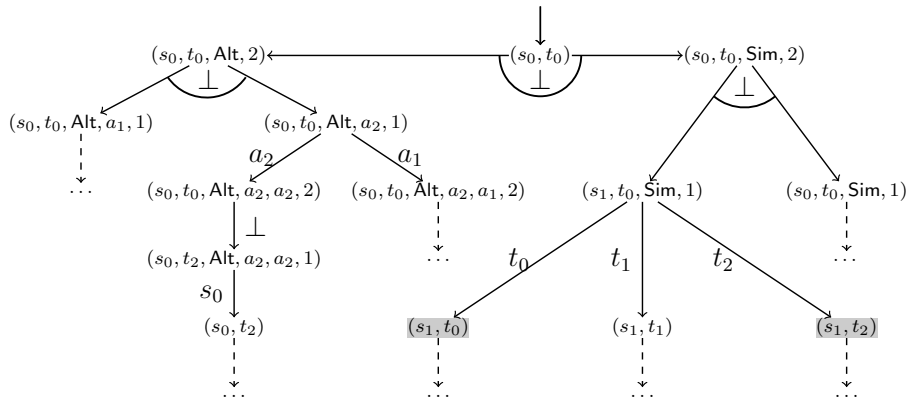

Figure 5.2: Part of the combined-simulation game of G and G' from Figure 5.1.

Theorem 5.1. *For all games G and G' we have $\mathcal{C}_{\max} = [\![\langle\langle 1 \rangle\rangle(\Box\neg p)]\!]_{G^{\mathcal{C}}} \cap (S \times S')$.*

Proof. The statement follows directly from the definition of combined simulation, and the fact that the game construction mimics the definition of combined simulation (as in the case of simulation and alternating simulation [148; 21; 149]). $\Box$

Winning strategies Given a combined-simulation game $G^{\mathcal{C}}$ we say that a strategy σ for the proponent is *winning* from a state s if for all strategies θ of the adversary we have $\mathsf{Play}(s, \sigma, \theta) \models \Box(\neg p)$. A strategy θ for the adversary is *winning* from state s if for all strategies σ of the proponent we have $\mathsf{Play}(s, \sigma, \theta) \models \mathsf{true}\,\mathcal{U}p$. Whenever the proponent (resp. adversary) has a winning strategy, the proponent (resp. adversary) also has memoryless winning strategy [49].

Combined simulation logical characterization Our next goal is to establish that combined simulation gives the logical characterization of C-ATL* and C-ATL. To prove the result we first introduce the following relation between plays: Given two plays $\rho = s_0 a_0 s_1 a_1 s_2 \cdots$ and $\rho' = s'_0 a'_0 s'_1 a'_1 s'_2 \cdots$ we write $\rho \leqslant_{\mathcal{C}} \rho'$ if for all $i \geq 0$ we have $(s_i, s'_i) \in \mathcal{C}_{\max}$.

Lemma 5.1. *Given two games G and G', let $\mathcal{C}_{\max}$ be the combined simulation. For all $(s, s') \in \mathcal{C}_{\max}$ the following assertions hold:*

- *For all Player 1 strategies σ in G, there exists a Player 1 strategy σ' in G' such that for every play $\rho' \in \mathsf{Plays}(s', \sigma')$ there exists a play $\rho \in \mathsf{Plays}(s, \sigma)$ such that $\rho \leqslant_{\mathcal{C}} \rho'$.*
- *For all pair of strategies σ and θ in G, there exists a pair of strategies σ' and θ' in G' such that $\mathsf{Play}(s, \sigma, \theta) \leqslant_{\mathcal{C}} \mathsf{Play}(s', \sigma', \theta')$,*

Proof. We present the details of the first item.

- Consider a winning strategy $\sigma^{\mathcal{C}}$ for the proponent in $G^{\mathcal{C}}$ such that for all $(s, s') \in \mathcal{C}_{\max}$ and against all strategies $\theta^{\mathcal{C}}$ we have $\mathsf{Play}(s, \sigma^{\mathcal{C}}, \theta^{\mathcal{C}}) \in [\![\Box(\neg p)]\!]$. Given the Player 1 strategy σ in G we construct σ' in G' using the strategy $\sigma^{\mathcal{C}}$. Consider a history $w \cdot s$ in G and $w' \cdot s' \in G'$ such that $(s, s') \in \mathcal{C}_{\max}$. Let $\sigma(w \cdot s) = a$. We define $\sigma'(w' \cdot s')$ as follows. Let h be an arbitrary history in $G^{\mathcal{C}}$ that only visits states in $\mathcal{C}_{\max}$ and ends in (s, s'). Let $a' = \sigma^{\mathcal{C}}(h \cdot (s, s', \mathsf{Alt}, 2) \cdot (s, s', \mathsf{Alt}, a, 2))$; (i.e., the action played by the strategy $\sigma^{\mathcal{C}}$ in response to the choice of checking alternating simulation and the action a by Player 2 in $G^{\mathcal{C}}$). Then the strategy σ' plays accordingly, i.e., $\sigma'(w' \cdot s') = a'$. In the next step for every choice t' of the adversary there exists a choice t of the proponent such that $L(t) = L'(t')$ and $(t, t') \in \mathcal{C}_{\max}$ and the matching can proceed.

- The proof is similar to the first item, and instead of using the step-wise alternating-simulation gadget for strategy construction (of the first item) we use the step-wise simulation gadget from $G^{\mathcal{C}}$ to construct the strategy pairs.

The desired result follows.

□

In the following theorem we establish the relation between combined simulation and the C-ATL* fragment of ATL*.

Theorem 5.2. *For all games G and G' we have $\mathcal{C}_{\max} = \preccurlyeq_C^* = \preccurlyeq_C$.*

Proof. First implication We first prove the implication $\mathcal{C}_{\max} \subseteq \preccurlyeq_C^*$. We will show the following assertions:

- For all states s and s' such that $(s, s') \in \mathcal{C}_{\max}$, we have that every C-ATL* state formula satisfied in s is also satisfied in s'.

- For all plays ρ and ρ' such that $\rho \leqslant_{\mathcal{C}} \rho'$, we have that every C-ATL* path formula satisfied in ρ is also satisfied in ρ'.

We will prove the theorem by induction on the structure of the formulas. The interesting cases for the induction step are formulas $\langle\langle 1 \rangle\rangle(\varphi)$ and $\langle\langle 1, 2 \rangle\rangle(\varphi)$, where φ is a path formula.

- Assume $s \models \langle\langle 1 \rangle\rangle(\varphi)$ and $(s, s') \in \mathcal{C}_{\max}$. It follows that there exists a strategy $\sigma \in \Sigma$ that ensures the path formula φ from state s against any strategy $\theta \in \Theta$. We want to show that $s' \models \langle\langle 1 \rangle\rangle(\varphi)$. By Lemma 5.1(item 1) we have that there exists a strategy σ' for Player 1 from s' such that for every play $\rho' \in \mathsf{Plays}(s', \sigma')$ there exists a play $\rho \in \mathsf{Plays}(s, \sigma)$ such that $\rho \leqslant_{\mathcal{C}} \rho'$. By inductive hypothesis we have that $s' \models \langle\langle 1 \rangle\rangle(\varphi)$.

- Assume $s \models \langle\langle 1, 2 \rangle\rangle(\varphi)$ and $\mathcal{C}(s, s')$. It follows that there exist strategies $\sigma \in \Sigma, \theta \in \Theta$ that ensure the path formula φ from state s. By Lemma 5.1(item 2) we have that there exist strategies σ' and θ' such that the two plays $\rho' = \mathsf{Play}(s', \sigma', \theta')$ and $\rho = \mathsf{Play}(s, \sigma, \theta)$ satisfy $\omega \leqslant_{\mathcal{C}} \omega'$. By inductive hypothesis we have that $s' \models \langle\langle 1, 2 \rangle\rangle(\varphi)$.

- Consider a path formula φ. If $\rho \leqslant_{\mathcal{C}} \rho'$, then by inductive hypothesis for every sub-formula φ' of φ we have that if $\rho \models \varphi'$ then $\rho' \models \varphi'$. It follows that if $\rho \models \varphi$ then $\rho' \models \varphi$.

Second implication It remains to prove the second implication $\preccurlyeq^*_C \subseteq \preccurlyeq_C \subseteq \mathcal{C}_{\max}$. Assume that given states s and s' we have that $(s, s') \notin \mathcal{C}_{\max}$, then there exists a winning strategy in the corresponding combined-simulation game for the adversary from state (s, s'), i.e., there exists a strategy $\theta^{\mathcal{C}}$ such that against all strategies $\sigma^{\mathcal{C}}$ we have $\mathsf{Play}((s, s'), \sigma^{\mathcal{C}}, \theta^{\mathcal{C}})$ reaches a state labeled p. As memoryless strategies are sufficient for both players in $G^{\mathcal{C}}$ [49], there also exists a bound $i \in \mathbb{N}$, such that the proponent fails to match the choice of the adversary in at most i turns. We sketch the inductive proof that there exists a formula with i nested operators $\langle\langle 1 \rangle\rangle\bigcirc$ or $\langle\langle 1, 2 \rangle\rangle\bigcirc$ that is satisfied in s but not in s'. For i equal to 0 the states can be distinguished by atomic propositions. For the inductive step one can express the simulation turns by a $\langle\langle 1, 2 \rangle\rangle(\bigcirc \ldots)$ formula and alternating simulation turns by a $\langle\langle 1 \rangle\rangle(\bigcirc \ldots)$ formula. It follows that $(s, s') \notin \preccurlyeq_C$. The result follows.

□

Note that in most cases the action set is constant and the state space of the games are huge. Then the combined simulation game construction is quadratic, and solving safety games on them can be achieved in linear time (in the size of the game) using discrete graph algorithms [150; 151].

Theorem 5.3. *Given two-player games G and G', the $\mathcal{C}_{\max}$, $\preccurlyeq^*_C$, and $\preccurlyeq_C$ relations can be computed in quadratic time using discrete graph algorithms.*

5.5 Qualitative Logics for Markov Decision Processes

In this section we consider Markov decisions processes (MDPs) and logics to reason qualitatively about them. We consider MDPs which can be viewed as a variant of two-player games defined in Section 2.2.4.

Interpretations We interpret an MDP in two distinct ways: (i) as a $1\frac{1}{2}$-player game and (ii) as a turn-based two-player game, where we regard the probabilistic states as Player-2 states. The $1\frac{1}{2}$-player interpretation is the classical view and it is explained in Section 2.2.4. We will use the two-player interpretation to relate logical characterizations of MDPs and logical characterization of two-player games with fragments of ATL^*.

Two-player interpretation The two-player interpretation corresponds to turn-based two-player games introduced in Section 2.1.4, where the probabilistic aspect of the MDP is replaced by a second player. Formally, given an MDP $G = (S, (S_1, S_P), A, \mathsf{Av}, \delta_1, \mathbf{P}, L, s_0)$ we define a turn-based two-player game $\widehat{G} = (\widehat{S}, \widehat{A}, \widehat{\mathsf{Av}}, \widehat{\delta}, \widehat{L}, \widehat{s_0})$ as follows: (i) the states are $\widehat{S} = S$; (ii) the set of actions contains a new action $\perp$ not present in A, i.e., $\widehat{A} = A \cup \{\perp\}$; (iii) the action-available function for states $s \in S_1$ is defined as $\widehat{\mathsf{Av}}(s) = \mathsf{Av}(s)$ and for states $s_p \in S_P$ as $\widehat{\mathsf{Av}}(s_p) = \{\perp\}$; (iv) for $s \in S_1$ and a in $\widehat{\mathsf{Av}}(s)$ we have $\widehat{\delta}(s, a) = \{\delta_1(s, a)\}$, and for $s_p \in S_P$ we have $\widehat{\delta}(s_p, \perp) = \mathrm{Supp}(\mathbf{P}(s_p))$; (v) the labeling function for a Player-1 state s is $\widehat{L}(s) = L(s) \cup \{\mathsf{turn}\}$ and for a Player-2 state s' coincides with $L(s')$; and (vi) the initial state is the same $\widehat{s_0} = s_0$. Given an MDP G we denote by $\widehat{G}$ the two-player interpretation of the MDP. Note that for all Player-1 states $s \in S_1$ we have $|\widehat{\delta}(s)| = 1$ and for all Player-2 states $s_p \in S_P$ we have $|\mathsf{Av}(s_p)| = 1$. Therefore for any MDP the corresponding two-player interpretation is a turn-based game.

Example 5.3. *In Figure 5.3 we present three MDPs G_1, G_2, and G' that we use as running examples. We thoroughly describe only MDP $G' = (S, (S_1, S_P), A, \mathsf{Av}, \delta_1, \mathbf{P}, L, s_0)$. Player-1 states, depicted as circles, are $S_1 = \{s'_0, s'_2, s'_3\}$ and probabilistic states, depicted as rectangles, are $S_P = \{s'_1, s'_4\}$. The set of actions is $A = \{a, b\}$. Action a is available in states s'_0, s'_2 and action b is available only in states s'_0, s'_3. The deterministic transition function is $\delta_1(s'_0, a) = s'_1, \delta_1(s'_0, b) = s'_4, \delta_1(s'_2, a) = s'_4, \delta_1(s'_2, b) = s'_4, \delta_1(s'_3, b) = s'_4$. The probabilistic transition function $\mathbf{P}$ gives the following probability distributions over possible successor states:*

$\mathbf{P}(s'_1, s'_2) = \frac{1}{2}, \mathbf{P}(s'_1, s'_3) = \frac{1}{2}, \mathbf{P}(s'_4, s'_3) = 1$. *There is a single atomic proposition* $p \in Ap$ *and the states labeled by* p *are depicted in gray. The initial state is* s'_0.

Parallel composition of MDPs An MDP is said to be alternating if the initial state is a Player-1 state, all the successors of Player-1 states are probabilistic states, and vice versa.

Definition 5.6 (Parallel composition of MDPs)**.** *Given two alternating MDPs*

$$G = (S, (S_1, S_P), A, \mathsf{Av}, \delta_1, \mathbf{P}, L, s_0)$$
$$G' = (S', (S'_1, S'_P), A, \mathsf{Av}', \delta'_1, \mathbf{P}', L', s'_0)$$

the parallel composition is an MDP

$$G \parallel G' = (\overline{S}, (\overline{S}_1, \overline{S}_P), A, \overline{\mathsf{Av}}, \overline{\delta}_1, \overline{\mathbf{P}}, \overline{L}, \overline{s}_0)$$

defined as follows:

1. *The states are* $\overline{S} = \overline{S}_1 \cup \overline{S}_P$*, where* $\overline{S}_1 = S_1 \times S'_1$ *and* $\overline{S}_P = S_P \times S'_P$.
2. *For a state* $(s, s') \in \overline{S}_1$ *we have* $\overline{\mathsf{Av}}((s, s')) = \mathsf{Av}(s) \cap \mathsf{Av}'(s')$.
3. *For a state* $(s, s') \in \overline{S}_1$ *and an action* $a \in \overline{\mathsf{Av}}((s, s'))$ *we have* $\overline{\delta}_1((s, s'), a) = (\delta_1(s, a), \delta'_1(s', a))$*;*
4. *For a state* $(s_p, s'_p) \in \overline{S}_P$ *we have* $\overline{\mathbf{P}}((s_p, s'_p), (t, t')) = \mathbf{P}(s_p, t) \cdot \mathbf{P}'(s'_p, t')$.
5. *For a state* $(s, s') \in \overline{S}$ *we have* $\overline{L}((s, s')) = L(s) \cup L'(s')$.
6. *The initial state is* (s_0, s'_0).

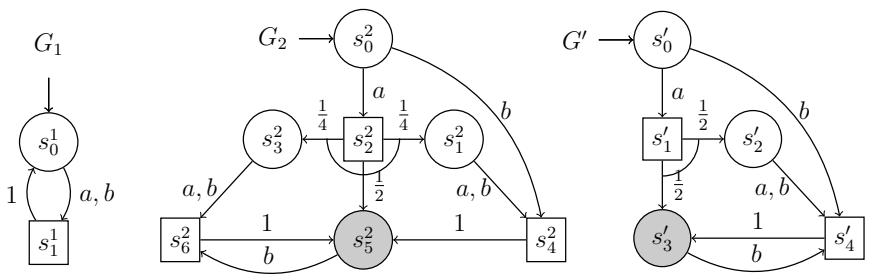

Figure 5.3: Examples of MDPs.

5.5.1 Qualitative Logics for MDPs

We consider the qualitative fragment of pCTL^* (see Section 2.2.5) and refer to the logic as *qualitative pCTL** (denoted as QCTL^*) as it can express qualitative properties of MDPs.

Syntax and semantics The syntax of the logic is given in positive normal form and is similar to the syntax of ATL^*. It has the same state and path formulas as ATL^* with the exception of path quantifiers. The logic QCTL^* comes with two path quantifiers (PQ), namely $\langle\mathsf{Almost}\rangle$ and $\langle\mathsf{Positive}\rangle$ (instead of $\langle\langle 1\rangle\rangle$, $\langle\langle 2\rangle\rangle$, $\langle\langle 1,2\rangle\rangle$, and $\langle\langle\emptyset\rangle\rangle$).

$$\mathrm{QCTL}^* \text{ path quantifiers:} \qquad \langle\mathsf{Almost}\rangle, \langle\mathsf{Positive}\rangle.$$

The semantics of the logic QCTL^* is the same for the fragment shared with ATL^*, therefore we only give semantics for the new path quantifiers. Given a path formula φ, we denote by $[\![\varphi]\!]_G$ the set of plays ρ such that $\rho \models \varphi$. For a state s and a path formula φ we have:

$$\begin{aligned} s &\models \langle\mathsf{Almost}\rangle(\varphi) && \text{iff } \exists\sigma \in \Sigma : \mathbb{P}_s^\sigma([\![\varphi]\!]) = 1 \\ s &\models \langle\mathsf{Positive}\rangle(\varphi) && \text{iff } \exists\sigma \in \Sigma : \mathbb{P}_s^\sigma([\![\varphi]\!]) > 0. \end{aligned}$$

As before, we denote by QCTL the fragment of QCTL^* where every temporal operator is immediately preceded by a path quantifier, and for a state formula ψ the set $[\![\psi]\!]_G$ denotes the set of states in G that satisfy the formula ψ.

Logical relation induced by QCTL **and** QCTL^* Given two MDPs G and G', the logical relation induced by QCTL^*, denoted as $\preccurlyeq_Q^*$, (resp. by QCTL, denoted as $\preccurlyeq_Q$), is defined as follows:

$$\preccurlyeq_Q^* = \{(s,s') \in S \times S' \mid \forall\psi \in \mathrm{QCTL}^* : \text{ if } s \models \psi \text{ then } s' \models \psi\}$$

(resp. $\forall\psi \in \mathrm{QCTL}$).

5.6 Characterization of Qualitative Simulation

In this section we establish the equivalence of the $\preccurlyeq_Q^*$ relation on MDPs with the $\preccurlyeq_C^*$ relation on the two-player interpretation of MDPs, i.e., we prove that for all MDPs G and G' we have $\preccurlyeq_Q^*(G,G') = \preccurlyeq_C(\widehat{G},\widehat{G}')$, where $\widehat{G}$ (resp. $\widehat{G}'$) is the two-player interpretation of the MDP G

(resp. G'). In the first step we show how to translate some of the QCTL formulas into C-ATL formulas. We only need to translate the path quantifiers due to the similarity of path formulas in the logics.

Lemma 5.2. *For all atomic propositions q, r and for all MDPs G, we have:*

$$[\![\langle \mathsf{Almost} \rangle (\bigcirc q)]\!]_G = [\![\langle\langle 1 \rangle\rangle (\bigcirc q)]\!]_{\widehat{G}} \tag{5.1}$$

$$[\![\langle \mathsf{Almost} \rangle (q \mathcal{W} r)]\!]_G = [\![\langle\langle 1 \rangle\rangle (q \mathcal{W} r)]\!]_{\widehat{G}} \tag{5.2}$$

$$[\![\langle \mathsf{Positive} \rangle (\bigcirc q)]\!]_G = [\![\langle\langle 1, 2 \rangle\rangle (\bigcirc q)]\!]_{\widehat{G}} \tag{5.3}$$

$$[\![\langle \mathsf{Positive} \rangle (q \mathcal{U} r)]\!]_G = [\![\langle\langle 1, 2 \rangle\rangle (q \mathcal{U} r)]\!]_{\widehat{G}} \tag{5.4}$$

Proof. Point 1 The inclusion $[\![\langle \mathsf{Almost} \rangle (\bigcirc q)]\!] \supseteq [\![\langle\langle 1 \rangle\rangle (\bigcirc q)]\!]$ follows from the fact that there exists a strategy for Player 1 such that for all strategies of Player 2 the next state reached satisfies q. It follows that the same strategy for Player 1 ensures the formula with probability 1. For the second inclusion $[\![\langle \mathsf{Almost} \rangle (\bigcirc q)]\!] \subseteq [\![\langle\langle 1 \rangle\rangle (\bigcirc q)]\!]$ we consider two cases: (i) let $s \in [\![\langle \mathsf{Almost} \rangle (\bigcirc q)]\!]$ be a Player-1 state. Then there exists an available action a that leads to a state that satisfies formula q. As s is a Player-1 state, the transition function under a has a unique successor. Therefore, playing the same action ensures q also in the two-player interpretation. The second case is that s is a probabilistic state. In that case all the successors in the support of the probabilistic transition function satisfy q. Therefore formula q is also satisfied in the two-player interpretation.

Point 2 As for the previous point the inclusion $[\![\langle \mathsf{Almost} \rangle (q \mathcal{W} r)]\!] \supseteq [\![\langle\langle 1 \rangle\rangle (q \mathcal{W} r)]\!]$ follows easily from the definition. For the second inclusion let σ be a strategy that satisfies the formula $q \mathcal{W} r$ almost-surely in the $1\frac{1}{2}$-player interpretation. Assume towards contradiction that there exists a strategy θ for Player 2 in the two-player interpretation, such that the play $\mathsf{Play}(s, \sigma, \theta)$ violates $q \mathcal{W} r$. It follows, that the play $\mathsf{Play}(s, \sigma, \theta)$ satisfies $\neg r \, \mathcal{U} \neg q$. This is possible only if there exists a finite path to a $\neg q$ state that uses only $\neg r$ states, and the finite path has a positive probability in the $1\frac{1}{2}$-player interpretation of the MDP. The contradiction follows.

Point 3. and 4 Point 3 follows similarly to Point 1, and Point 4 follows the same arguments as in Point 2.

□

Lemma 5.3. *For all atomic propositions r and for all MDPs G we have:* $[\![\langle \mathsf{Positive} \rangle (\Box\, r)]\!]_G = [\![\langle \mathsf{Positive} \rangle (r \, \mathcal{U} \langle \mathsf{Almost} \rangle (\Box\, r))]\!]_G$.

Proof. The result follows from [136, Lemma 1] (shown even for a more general class of partially observable MDPs) and from [152] showing that deterministic strategies are sufficient in POMDPs.

□

Lemma 5.4. *For all atomic propositions q, r and for all MDPs G, we have:*

$$[\![\langle \mathsf{Positive} \rangle (q \mathcal{W} r)]\!]_G = [\![\langle\langle 1,2 \rangle\rangle (q \mathcal{U} r)]\!]_{\widehat{G}} \cup [\![\langle\langle 1,2 \rangle\rangle (q \mathcal{U} (\langle\langle 1 \rangle\rangle (q \mathcal{W} \mathsf{false})))]\!]_{\widehat{G}}$$

Proof. By definition we have that $[\![\langle \mathsf{Positive} \rangle (q \mathcal{W} r)]\!] = [\![\langle \mathsf{Positive} \rangle ((q \mathcal{U} r) \vee (\Box q))]\!]$. We write the formula as follows: $[\![\langle \mathsf{Positive} \rangle ((q \mathcal{U} r) \vee (\Box q))]\!] = [\![\langle \mathsf{Positive} \rangle (q \mathcal{U} r)]\!] \cup [\![\langle \mathsf{Positive} \rangle (\Box q)]\!]$. By Lemma 5.3 we have that $[\![\langle \mathsf{Positive} \rangle (\Box q)]\!] = [\![\langle \mathsf{Positive} \rangle (q \mathcal{U} \langle \mathsf{Almost} \rangle (\Box\, q))]\!]$. Note that $\Box\, q \equiv q \mathcal{W} \mathsf{false}$. All these facts together with the already established translations presented in Lemma 5.2 give us the desired result.

□

To complete the translation of temporal operators it remains to express the QCTL formula $[\![\langle \mathsf{Almost} \rangle (q \mathcal{U} r)]\!]$ in terms of C-ATL. We first introduce the Apre function:

Apre Given two sets of states $X, Y \subseteq S$ we define the predecessor operator Apre as follows:

$$\begin{aligned} \mathsf{Apre}(Y, X) \;=\; & \{s \in S_1 \mid \exists a \in \mathsf{Av}(s) \;:\; \delta_1(s,a) \in X \cap Y\} \cup \\ & \{s_p \in S_P \mid \mathrm{Supp}(\mathbf{P}(s_p)) \subseteq Y \wedge \mathrm{Supp}(\mathbf{P}(s_p)) \cap X \neq \emptyset\}. \end{aligned}$$

Intuitively, in the $1\frac{1}{2}$ interpretation the Apre function given two sets of states X and Y, selects Player-1 states from which Player 1 can enforce the next state to be in $X \cap Y$, and selects probabilistic states such that the next state is with probability one in Y and with positive probability in X. As is shown in [126] we can express the states $[\![\langle \mathsf{Almost} \rangle (q \mathcal{U} r)]\!]$ using the following μ-calculus notation, where μ (resp. ν) denotes the least (resp. greatest) fixpoint:

$$[\![\langle \mathsf{Almost} \rangle (q \mathcal{U} r)]\!] = \nu Y.\mu X.([\![r]\!] \cup ([\![q]\!] \cap \mathsf{Apre}(Y, X))) \tag{5.5}$$

The fixpoint computation on an MDP with n states can be described as follows: the variable Y_0 is initialized to whole state space, and in each iteration i the variable $X_{i,0}$ is initialized to the empty set; the variable $X_{i,j+1}$ is computed from $X_{i,j}$ applying the one step Apre operator. Finally, the variable Y_i is set as the fixpoint of iteration i.

Formally, for $1 \leq i \leq n$ and $0 \leq j \leq n-1$ we have

$$Y_0 = [\![\mathsf{true}]\!]; \quad X_{i,0} = [\![\mathsf{false}]\!]; \quad X_{i,j+1} = ([\![r]\!] \cup ([\![q]\!] \cap \mathsf{Apre}(Y_{i-1}, X_{i,j}))); \quad Y_i = X_{i,n};$$

and then $Y_n = [\![\langle \mathsf{Almost} \rangle (q \mathcal{U} r)]\!]$. Next we show that the Apre function can be expressed in C-ATL. For C-ATL formulas ψ_1, ψ_2 we define:

$$F_{\mathsf{Apre}}(\psi_1, \psi_2) = \langle\langle 1 \rangle\rangle (\bigcirc \psi_1) \wedge \langle\langle 1, 2 \rangle\rangle (\bigcirc (\psi_1 \wedge \psi_2))$$

Lemma 5.5. *For all MDPs G and* C-ATL *state formulas ψ_1, ψ_2 we have:*

$$[\![F_{\mathsf{Apre}}(\psi_1, \psi_2)]\!]_{\widehat{G}} = \mathsf{Apre}([\![\psi_1]\!]_{\widehat{G}}, [\![\psi_2]\!]_{\widehat{G}})$$

Proof. We prove the two inclusions. We start with $\mathsf{Apre}([\![\psi_1]\!], [\![\psi_2]\!]) \subseteq [\![F_{\mathsf{Apre}}(\psi_1, \psi_2)]\!]$. Let s be a state in $\mathsf{Apre}([\![\psi_1]\!], [\![\psi_2]\!])$, we consider two cases: (i) $s \in S_1$; and (ii) $s \in S_P$. For the case (i) it follows from the definition of Apre that there exists an action $a \in \mathsf{Av}(s)$ such that the unique state $\delta_1(s, a)$ satisfies $\psi_1 \wedge \psi_2$. It follows that $s \in [\![\langle\langle 1 \rangle\rangle (\bigcirc \psi_1) \wedge \langle\langle 1, 2 \rangle\rangle (\bigcirc (\psi_1 \wedge \psi_2))]\!]$ and therefore $s \in [\![F_{\mathsf{Apre}}(\psi_1, \psi_2)]\!]$. In case (ii) $s \in S_P$, we have by definition $\mathrm{Supp}(\mathbf{P}(s)) \subseteq [\![\psi_1]\!]$, and $\mathrm{Supp}(\mathbf{P}(s)) \cap [\![\psi_2]\!] \neq \emptyset$. It follows that $\mathrm{Supp}(\mathbf{P}(s)) \cap [\![\psi_1 \wedge \psi_2]\!] \neq \emptyset$ and $s \in [\![\langle\langle 1 \rangle\rangle (\bigcirc \psi_1) \wedge \langle\langle 1, 2 \rangle\rangle (\bigcirc (\psi_1 \wedge \psi_2))]\!]$, and therefore $s \in [\![F_{\mathsf{Apre}}(\psi_1, \psi_2)]\!]$.

We continue with the second inclusion $[\![F_{\mathsf{Apre}}(\psi_1, \psi_2)]\!] \subseteq \mathsf{Apre}([\![\psi_1]\!], [\![\psi_2]\!])$. Let s be a state in $[\![F_{\mathsf{Apre}}(\psi_1, \psi_2)]\!]$, we again consider two cases: (i) $s \in S_1$; and (ii) $s \in S_P$. For case (i) when $s \in S_1$ assume $s \in [\![\langle\langle 1 \rangle\rangle (\bigcirc \psi_1) \wedge \langle\langle 1, 2 \rangle\rangle (\bigcirc (\psi_1 \wedge \psi_2))]\!]$, it follows that there exists an available action $a \in \mathsf{Av}(s)$ such that the unique state $\delta_1(s, a)$ is in $[\![\psi_1 \wedge \psi_2]\!]$. For the second case (ii) when $s \in S_P$ we again assume $s \in [\![\langle\langle 1 \rangle\rangle (\bigcirc \psi_1) \wedge \langle\langle 1, 2 \rangle\rangle (\bigcirc (\psi_1 \wedge \psi_2))]\!]$. The first part of the formula ensures that $\mathbf{P}(s) \subseteq [\![\psi_1]\!]$ and the second part ensures that $\mathbf{P}(s) \cap [\![\psi_1 \wedge \psi_2]\!] \neq \emptyset$. The desired result follows.

□

The following lemma shows the first of the two inclusions:

Lemma 5.6. *For all MDPs G and G' we have $\preccurlyeq_C (\widehat{G}, \widehat{G}') \subseteq \preccurlyeq_Q (G, G')$.*

Proof. We prove the counterpositive, i.e., we construct a mapping of formulas $f : \mathrm{QCTL} \to$ C-ATL such that given two states s, s' and a QCTL formula ψ we have that if $s \models \psi$ and $s' \not\models \psi$ then the C-ATL formula $f(\psi)$ is true in s and not true in s'. We proceed by structural induction

on the QCTL formula and replace parts that are in scope of a path quantifier by their C-ATL version. The cases where ψ is an atomic proposition or a Boolean combination of formulas are straightforward. It remains to translate the formulas $\langle\mathsf{Almost}\rangle(\bigcirc\varphi_1)$, $\langle\mathsf{Almost}\rangle(\varphi_1\mathcal{W}\varphi_2)$, and $\langle\mathsf{Almost}\rangle(\varphi_1\mathcal{U}\varphi_2)$ for QCTL formulas φ_1, φ_2. The translation of the first two follows directly from Lemma 5.2, therefore it remains to translate the QCTL formula $\langle\mathsf{Almost}\rangle(\varphi_1\mathcal{U}\varphi_2)$. We proceed by encoding the fixpoint computation of the $\langle\mathsf{Almost}\rangle(\varphi_1\mathcal{U}\varphi_2)$ formula into nested C-ATL formulas. Let n be the number of states of the MDP. Let $\{\widetilde{\phi}_i,\ \phi_{i,j} \mid 0 \leq i, j \leq n\}$ be a set of formulas defined by the following clauses:

$$\begin{aligned}
& \widetilde{\phi}_0 = \mathsf{true}; \\
\forall 1 \leq i \leq n: \quad & \phi_{i,0} = \mathsf{false} \\
\forall 1 \leq i \leq n . \forall 0 \leq j \leq n-1: \quad & \phi_{i,j+1} = f(\varphi_2) \vee (f(\varphi_1) \wedge F_{\mathsf{Apre}}(\widetilde{\phi}_{i-1}, \phi_{i,j})) \\
\forall 1 \leq i \leq n: \quad & \widetilde{\phi}_i = \phi_{i,n};
\end{aligned}$$

By Lemma 5.5 the set of nested formulas $\phi_{i,j}$ represents the computation of $X_{i,j}$ and $\widetilde{\phi}_i$ the computation of Y_i (for the computation of the fixpoint formula). It follows that we have $[\![\langle\mathsf{Almost}\rangle(\varphi_1\mathcal{U}\varphi_2)]\!] = [\![\widetilde{\phi}_n]\!]$ and concludes the translation. The translation for formulas $\langle\mathsf{Positive}\rangle(\bigcirc\varphi_1)$, $\langle\mathsf{Positive}\rangle(\varphi_1\mathcal{W}\varphi_2)$, and $\langle\mathsf{Positive}\rangle(\varphi_1\mathcal{U}\varphi_2)$ to C-ATL formulas follows from Lemma 5.2 and Lemma 5.4. The desired result follows.

□

Lemma 5.7. *For all MDPs G and G' we have $\preccurlyeq_Q (G, G') \subseteq \preccurlyeq_C (\widehat{G}, \widehat{G}')$.*

Proof. Given MDPs with n states in total, it follows from the proof of Theorem 5.2 for the combined-simulation game that the n-step approximation $\preccurlyeq_C^n$ is exactly the same as $\preccurlyeq_C$. We define a sequence $\Psi_0, \Psi_1, \ldots, \Psi_n$ of sets of formulas of QCTL with the property that $s \preccurlyeq_C^i t$ iff every formula $\psi \in \Psi_i$ that is true in s is also true in t. We denote by $\mathsf{BoolC}(\Psi)$ all the formulas that consist of disjunctions and conjunctions of formulas in Ψ. We assume that $\mathsf{BoolC}(\Psi)$ does not contain repeated elements, therefore from finiteness of Ψ follows finiteness of $\mathsf{BoolC}(\Psi)$. We define $\Psi_0 = \mathsf{BoolC}(\{q, \neg q \mid q \in Ap\})$, and for all $0 \leq i < n$ we define $\Psi_{i+1} = \mathsf{BoolC}(\{\Psi_i \cup \{\langle\mathsf{Positive}\rangle(\bigcirc\psi), \langle\mathsf{Almost}\rangle(\bigcirc\psi) \mid \psi \in \Psi_i\}\})$. The formulas in $\Psi_0, \Psi_1, \ldots, \Psi_n$ provide witnesses that for all $0 \leq i \leq n$ we have that $\preccurlyeq_Q \subseteq \preccurlyeq_C^i$, in particular we have that $\preccurlyeq_Q \subseteq \preccurlyeq_C$.

□

Theorem 5.4. *For all MDPs G and G' we have* $\preccurlyeq_Q (G, G') = \preccurlyeq_C (\widehat{G}, \widehat{G'})$.

Theorem 5.5. *For all MDPs G and G' we have* $\preccurlyeq_Q^* (G, G') = \preccurlyeq_Q (G, G')$

Proof. We need to show that if a QCTL^* formula distinguishes two states, then there is a QCTL formula that also distinguishes them. The basic idea is similar to the proof of [153, Theorem 7.1, assertion 2]. We first construct a deterministic parity automata given the formula in QCTL^*, and the almost-sure or positive solutions for MDPs with parity objectives can be encoded as a μ-calculus formula [127]. The translation of μ-calculus formulas to a QCTL formula is done as in Lemma 5.6.

□

The size of the formulas when translating from QCTL^* to QCTL (Theorem 5.5) may be doubly exponential in the size of the input formula. Note, that the translation of LTL to deterministic parity automata is already doubly-exponential [154].

Theorem 5.6. *Given MDPs G and G' the relation* $\preccurlyeq_Q^* (G, G')$ *can be computed in quadratic time using discrete graph algorithms.*

Proof. Follows directly from Theorems 5.3, 5.4, and 5.5.

□

5.7 Counterexample Guided Abstraction Refinement for Combined Simulation

In this section we present a CEGAR approach for the computation of the combined simulation relation in two-player games.

5.7.1 Simulation Abstraction and Alternating-Simulation Abstraction

Abstraction An *abstraction* of a game consists of a partition of the game graph such that in each partition the atomic proposition labeling match for all states. Given an abstraction of a game, the abstract game can be defined by collapsing states of each partition and redefining the

action-available and transition functions. The redefinition of the action-available and transition functions can either increase or decrease the power of the players. If we increase the power of Player 1 and decrease the power of Player 2, then the abstract game will be in alternating simulation with the original game, and if we increase the power of both players, then the abstract game will simulate the original game. We now formally define the partitions, and the two abstractions.

Partitions for abstraction A *partition* of a game $G = (S, A, \mathsf{Av}, \delta, L, s_0)$ is an equivalence relation $\Pi = \{\pi_1, \pi_2, \ldots, \pi_k\}$ on S such that: (i) for all $1 \leq i \leq k$ we have $\pi_i \subseteq S$ and for all $s, s' \in \pi_i$ we have $L(s) = L(s')$ (labeling match); (ii) $\bigcup_{1 \leq i \leq k} \pi_i = S$ (covers the state space); and (iii) for all $1 \leq i, j \leq k$, such that $i \neq j$ we have $\pi_i \cap \pi_j = \emptyset$ (disjoint). Note that in turn-based games Player 1 and Player 2 states are distinguished by proposition turn, so they belong to different partitions.

Simulation abstraction Given a two-player game $G = (S, A, \mathsf{Av}, \delta, L, s_0)$ and a partition Π of G, we define the *simulation abstraction of* G as a two-player game $Abs^{\Pi}_{\mathcal{S}}(G) = (\overline{S}, A, \overline{\mathsf{Av}}, \overline{\delta}, \overline{L}, \overline{s}_0)$, where

- $\overline{S} = \Pi$: the partitions in Π are the states of the abstract game.
- For all $\pi_i \in \Pi$ we have $\overline{\mathsf{Av}}(\pi_i) = \bigcup_{s \in \pi_i} \mathsf{Av}(s)$: the set of available actions is the union of the actions available to the states in the partition, and this gives more power to Player 1.
- For all $\pi_i \in \Pi$ and $a \in \overline{\mathsf{Av}}(\pi_i)$ we have

  $$\overline{\delta}(\pi_i, a) = \{\pi_j \mid \exists s \in \pi_i : \ (a \in \mathsf{Av}(s) \wedge \exists s' \in \pi_j : \ s' \in \delta(s, a))\},$$

  i.e. there is a transition from a partition π_i given an action a to a partition π_j if some state $s \in \pi_i$ can make an a-transition to some state in $s' \in \pi_j$, and this gives more power to Player 2.
- For all $\pi_i \in \Pi$ we have $\overline{L}(\pi_i) = L(s)$ for some $s \in \pi_i$: the abstract labeling is well-defined, since all states in a partition are labeled by the same atomic propositions.
- $\overline{s}_0$ is the partition in Π that contains state s_0.

Alternating-simulation abstraction Given a two-player game $G = (S, A, \mathsf{Av}, \delta, L, s_0)$ and a partition Π of G, we define the *alternating-simulation abstraction of* G as a two-player game $Abs^{\Pi}_{\mathcal{A}}(G) = (\widetilde{S}, A, \widetilde{\mathsf{Av}}, \widetilde{\delta}, \widetilde{L}, \widetilde{s}_0)$, where

- $\widetilde{S} = \Pi$;
- For all $\pi_i \in \Pi$ we have $\widetilde{\mathsf{Av}}(\pi_i) = \bigcup_{s \in \pi_i} \mathsf{Av}(s)$;
- For all $\pi_i \in \Pi$ and $a \in \widetilde{\mathsf{Av}}(\pi_i)$ we have

$$\widetilde{\delta}(\pi_i, a) = \{\pi_j \mid \forall s \in \pi_i : \ (a \in \mathsf{Av}(s) \wedge \ \exists s' \in \pi_j : \ s' \in \delta(s, a))\},$$

  i.e. there is a transition from a partition π_i given an action a to a partition π_j if all states $s \in \pi_i$ can make an a-transition to some state in $s' \in \pi_j$, and this gives less power to Player 2. For technical convenience we assume $\widetilde{\delta}(\pi_i, a)$ is non-empty.
- For all $\pi_i \in \Pi$ we have $\widetilde{L}(\pi_i) = L(s)$ for some $s \in \pi_i$;
- $\widetilde{s}_0$ is the partition in Π that contains state s_0 (as in the case of simulation abstraction).

The following proposition states that the (alternating-)simulation abstraction of a game G is in (alternating-)simulation with G.

Proposition 5.3. *For all partitions Π of a two-player game G we have: (1) $G \leqslant_{\mathcal{A}} Abs^{\Pi}_{\mathcal{A}}(G)$; and (2) $G \leqslant_{\mathcal{S}} Abs^{\Pi}_{\mathcal{S}}(G)$.*

Example 5.4. *Consider a two-player interpretation $\widehat{G}_2$ of the MDP G_2 from Figure 5.3. The coarsest partition of $\widehat{G}_2$ is $\Pi = \{\pi_0, \pi_1, \pi_2\}$, where $\pi_0 = \{s^2_0, s^2_1, s^2_3\}, \pi_1 = \{s^2_2, s^2_4, s^2_6\}, \pi_2 = \{s^2_5\}$. The alternating-simulation abstraction and the simulation abstraction of Π are depicted in Figure 5.4.*

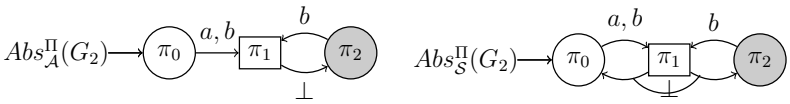

Figure 5.4: Alternating-simulation and simulation abstractions of the two-player interpretation $\widehat{G}_2$ (MDP G_2 from Figure 5.3).

5.7.2 Sound Assume-Guarantee Rule

In this section we present the sound assume-guarantee rule for the combined-simulation problem. To achieve this we first need an extension of the notion of combined-simulation game.

Definition 5.7 (Modified combined-simulation game)**.** *Consider games* $G^{\mathsf{Alt}} = (S, A, \delta^{\mathsf{Alt}}, \mathsf{Av}^{\mathsf{Alt}}, L, s_0)$, $G^{\mathsf{Sim}} = (S, A, \delta^{\mathsf{Sim}}, \mathsf{Av}^{\mathsf{Sim}}, L, s_0)$ *and* $G' = (S', A, \delta', \mathsf{Av}', L', s'_0)$. *The* modified simulation game $G^{\mathcal{M}} = (S^{\mathcal{M}}, A^{\mathcal{M}}, \mathsf{Av}^{\mathcal{M}}, \delta^{\mathcal{M}}, L^{\mathcal{M}}, s_0^{\mathcal{M}})$ *is defined exactly like the combined simulation game given* G^{Alt} *and* G'*, with the exception that the step-wise simulation gadget is defined using the transitions of* G^{Sim} *instead of* G^{Alt}*. Formally, we change the transitions as follows:*

- Checking step-wise simulation conditions *Transition (a) from Definition 5.5 is redefined: for a state* $(s, s', \mathsf{Sim}, 2)$ *we have only one action* $\perp$ *available for Player 1 and we have* $\delta^{\mathcal{M}}((s, s', \mathsf{Sim}, 2), \perp) = \{(t, s', \mathsf{Sim}, 1) \mid \exists a \in \mathsf{Av}^{\mathsf{Sim}}(s) : \ t \in \delta^{\mathsf{Sim}}(s, a)\}$.

We write $(G^{\mathsf{Alt}} \otimes G^{\mathsf{Sim}}) \leqslant_{\mathcal{M}} G'$ *if and only if* $(s_0, s'_0) \in [\![\langle\langle 1 \rangle\rangle(\Box\neg p)]\!]_{G^{\mathcal{M}}}$.

Proposition 5.4. *Let* $G, G', G^{\mathsf{Alt}}, G^{\mathsf{Sim}}$ *be games such that* $G \leqslant_{\mathcal{A}} G^{\mathsf{Alt}}$ *and* $G \leqslant_{\mathcal{S}} G^{\mathsf{Sim}}$*. Then* $(G^{\mathsf{Alt}} \otimes G^{\mathsf{Sim}}) \leqslant_{\mathcal{M}} G'$ *implies* $G \leqslant_{\mathcal{C}} G'$.

The key proof idea for the above proposition is as follows: if $G \leqslant_{\mathcal{A}} G^{\mathsf{Alt}}$ and $G \leqslant_{\mathcal{S}} G^{\mathsf{Sim}}$, then in the modified combined-simulation game $G^{\mathcal{M}}$ the adversary (Player 2) is stronger than in the combined-simulation game $G^{\mathcal{C}}$. Hence winning in $G^{\mathcal{M}}$ for the proponent (Player 1) implies winning in $G^{\mathcal{C}}$ and gives the desired result of the proposition.

Sound assume-guarantee method Given two games G_1 and G_2, checking whether their parallel composition $G_1 \parallel G_2$ is in combined simulation with a game G' can be done explicitly by constructing the synchronized product. The composition, however, may be much larger than the components and thus make the method ineffective in practical cases. We present an alternative method that proves combined simulation in a compositional manner, by abstracting G_2 with some partition Π and then composing it with G_1. The sound assume-guarantee rule follows from Proposition 5.3 and Proposition 5.4.

Proposition 5.5 (Sound assume-guarantee rule)**.** *Given games G_1, G_2, G', and a partition Π of G_2, let $\mathsf{A} = G_1 \parallel Abs^{\Pi}_{\mathcal{A}}(G_2)$ and $\mathsf{S} = G_1 \parallel Abs^{\Pi}_{\mathcal{S}}(G_2)$. If $(\mathsf{A} \otimes \mathsf{S}) \leqslant_{\mathcal{M}} G'$, then $(G_1 \parallel G_2) \leqslant_{\mathcal{C}} G'$, i.e.,*

$$\frac{\mathsf{A} = G_1 \parallel Abs^{\Pi}_{\mathcal{A}}(G_2); \qquad \mathsf{S} = G_1 \parallel Abs^{\Pi}_{\mathcal{S}}(G_2); \qquad (\mathsf{A} \otimes \mathsf{S}) \leqslant_{\mathcal{M}} G'}{(G_1 \parallel G_2) \leqslant_{\mathcal{C}} G'}. \tag{5.6}$$

Note that for the trivial partition Π, where every equivalence relation is a singleton, the modified combined-simulation game coincides with the combined simulation game. We will use this fact to argue about completeness of our CEGAR approach.

If the partition Π is coarse, then the abstractions in the assume-guarantee rule can be smaller than G_2 and also their composition with G_1. As a consequence, combined simulation can be faster as compared to explicitly computing the composition. In Section 5.7.4 we describe how to effectively compute the partitions Π and refine them using CEGAR approach.

5.7.3 Counterexamples Analysis

Representation of counterexamples If the premise $(\mathsf{A} \otimes \mathsf{S}) \leqslant_{\mathcal{M}} G'$ of the assume-guarantee rule (5.6) is not satisfied, then the adversary (Player 2) has a memoryless winning strategy θ_{abs} in $G^{\mathcal{M}}$, and the memoryless strategy is the *counterexample*. To use the sound assume-guarantee rule (5.6) in a CEGAR loop, we need analysis of counterexamples. Note that in $G^{\mathcal{M}}$ Player 2 has a reachability objective, and thus a winning strategy θ_{abs} ensures that the target set is always reached from the starting state, and hence no cycle can be formed without reaching the target state once the memoryless winning strategy is fixed. Hence we represent counterexamples as directed-acyclic graphs (given the strategy θ_{abs} we denote the corresponding directed acyclic graph as $\mathsf{DAG}(\theta_{\mathsf{abs}})$), where the leafs are the target states and every non-leaf state has a single successor chosen by the strategy of Player 2 and has all available actions for Player 1.

Abstract, concrete, and spurious counterexamples Given two-player games G_1 and G_2, let $G = (G_1 \parallel G_2)$ be the parallel composition. Given G and G', let $G^{\mathcal{C}}$ be the combined-simulation game of G and G'. The abstract game $G^{\mathcal{M}}$ is the modified combined-simulation game of $(\mathsf{A} \otimes \mathsf{S})$ and G', where $\mathsf{A} = G_1 \parallel Abs^{\Pi}_{\mathcal{A}}(G_2)$ and $\mathsf{S} = G_1 \parallel Abs^{\Pi}_{\mathcal{S}}(G_2)$. We refer to a counterexample θ_{abs} in $G^{\mathcal{M}}$ as *abstract*, and to a counterexample θ_{con} in $G^{\mathcal{C}}$ as *concrete*. An abstract counterexample θ_{abs} is *feasible* if we can substitute partitions in A and S in a rooted subtree of θ_{abs} with states of

G_2 to obtain a concrete counterexample (see [105] for details). An abstract counterexample is *spurious* if it is not feasible.

Concretization of counterexamples We follow the approach of [105] to check the feasibility of a counterexample by finding a *concretization function* Conc from states in $G^{\mathcal{M}}$ to a set of states in G_2 that witnesses a concrete strategy θ_{con} from the abstract strategy θ_{abs}. A state in $G^{\mathcal{M}}$ has as one of its components a subset of states $\pi_i \in \Pi$, which is an equivalence class from the abstracted game G_2. Intuitively, for a state $\overline{s}$ of $G^{\mathcal{M}}$ in the counterexample graph $\mathsf{DAG}(\theta_{\mathsf{abs}})$, the concretization function represents the subset of π_i where a concrete winning strategy exists that replays the strategy represented by the subtree of the graph $\mathsf{DAG}(\theta_{\mathsf{abs}})$ rooted at state $\overline{s}$.

Computation of the concretization function Given an abstract counterexample θ_{abs} and a state $\overline{s}$ in $G^{\mathcal{M}}$, let $\mathsf{Succ}(\overline{s})$ be the set of all successors of $\overline{s}$ in $G^{\mathcal{M}}$ given θ_{abs} is fixed by Player 2. The concretization function Conc is computed inductively on the structure of the abstract counterexample θ_{abs} starting from the leaves. The formal description of the concretization computation is given in Figure 5.5, where the concretization of a state $\overline{s}$ in the abstract counterexample is computed from its successors in the DAG. We use the notation Av^1, Av^2, and δ^2 to represent the action-available functions of G_1 and G_2, and the transition function of G_2, respectively.

Proposition 5.6. *[105, Proposition 2] An abstract counterexample θ_{abs} is feasible if and only if the concretization function Conc of the root of the graph $\mathsf{DAG}(\theta_{\mathsf{abs}})$ contains the initial state of the game G_2.*

Illustrative examples We present intuitive description of two representative cases of concretization from Figure 5.5: (1) Consider a state $\overline{s} = ((s_1, \pi_2), s', \mathsf{Alt}, 2)$ where the abstract counterexample chooses the successor $\overline{s}' = ((s_1, \pi_2), s', \mathsf{Alt}, a, 1)$ (intuitively this corresponds to choice of action a). The concretization $\mathsf{Conc}(\overline{s}) = \{s \in \pi_2 \mid a \in \mathsf{Av}^2(s) \wedge s \in \mathsf{Conc}(\overline{s}')\}$ is the subset of states in π_2 where the action a is available and s also belongs to the concretization of the successor state $\overline{s}'$. (2) For a state $\overline{s} = ((s_1, \pi_2), s', \mathsf{Alt}, a, a', 1)$, the concretization is the set of states where action a is not available or all successors given action a belong to the concretization of the successors of $\overline{s}$.

$$\overline{s} = ((s_1, \pi_2), s') \quad : \quad \mathsf{Conc}(\overline{s}) = \begin{cases} \pi_2 & \overline{s} \text{ is a leaf} \\ \mathsf{Conc}(\overline{s}') & \text{otherwise, where } \mathsf{Succ}(\overline{s}) = \{\overline{s}'\} \end{cases}$$

$$\overline{s} = ((s_1, \pi_2), s', \mathsf{Sim}, 2) \quad : \quad \mathsf{Conc}(\overline{s}) = \{s \in \pi_2 \mid \exists a \in \mathsf{Av}^1(s_1) \cap \mathsf{Av}^2(s) : \delta^2(s, a) \cap \mathsf{Conc}(\overline{s}') \neq \emptyset\}$$
$$\text{where } \mathsf{Succ}(\overline{s}) = \{\overline{s}'\}$$

$$\overline{s} = ((s_1, \pi_2), s', \mathsf{Sim}, 1) \quad : \quad \mathsf{Conc}(\overline{s}) = \bigcap_{\overline{s}' \in \mathsf{Succ}(\overline{s})} \mathsf{Conc}(\overline{s}')$$

$$\overline{s} = ((s_1, \pi_2), s', \mathsf{Alt}, 2) \quad : \quad \mathsf{Conc}(\overline{s}) = \{s \in \pi_2 \mid a \in \mathsf{Av}^2(s) \wedge s \in \mathsf{Conc}(\overline{s}'), \} \text{ where}$$
$$\mathsf{Succ}(\overline{s}) = \{\overline{s}'\} \text{ and } \overline{s}' = ((s_1, \pi_2), s', \mathsf{Alt}, 2, a)$$

$$\overline{s} = ((s_1, \pi_2), s', \mathsf{Alt}, a, 1) \quad : \quad \mathsf{Conc}(\overline{s}) = \bigcap_{\overline{s}' \in \mathsf{Succ}(\overline{s})} \mathsf{Conc}(\overline{s}')$$

$$\overline{s} = ((s_1, \pi_2), s', \mathsf{Alt}, a, a', 2) \quad : \quad \mathsf{Conc}(\overline{s}) = \mathsf{Conc}(\overline{s}'), \text{ where } \mathsf{Succ}(\overline{s}) = \{\overline{s}'\}$$

$$\overline{s} = ((s_1, \pi_2), s', \mathsf{Alt}, a, a', 1) \quad : \quad \mathsf{Conc}(\overline{s}) = \{s \in \pi_2 \mid a \notin \mathsf{Av}^2(s) \vee \delta^2(s, a) \subseteq \bigcup_{\overline{s}' \in \mathsf{Succ}(\overline{s})} \mathsf{Conc}(\overline{s}')\}$$

Figure 5.5: Concretization function; $\overline{s}$ is a state in an abstract counterexample.

Example 5.5. *Consider MDPs G_1, G_2, G' in Figure 5.3 interpreted as games and the abstract games $Abs^{\Pi}_{\mathcal{A}}(\widehat{G}_2)$, $Abs^{\Pi}_{\mathcal{S}}(\widehat{G}_2)$ in Figure 5.4. Let $\mathsf{A} = \widehat{G}_1 \parallel Abs^{\Pi}_{\mathcal{A}}(\widehat{G}_2)$ and $\mathsf{S} = \widehat{G}_1 \parallel Abs^{\Pi}_{\mathcal{S}}(\widehat{G}_2)$. Figure 5.6 shows part of an abstract counterexample to the modified combined-simulation game of $(\mathsf{A} \otimes \mathsf{S})$ and G'. In this counterexample the adversary first plays in the simulation gadget and the proponent responds by moving to a state $((s_1^1, \pi_1), s_1')$ or a state $((s_1^1, \pi_1), s_4')$ (their successors are not depicted in Figure 5.6). From the state $((s_1^1, \pi_1), s_1')$ the adversary has a winning strategy by playing in the alternating-simulation gadget, and from $((s_1^1, \pi_1), s_4')$ by playing in the simulation gadget. The dashed shows assign the concretization of states in the abstract counterexample. The counterexample is spurious, since the initial state of G_2 does not belong to the concretization of the initial state of the counterexample.*

5.7.4 CEGAR

The counterexample analysis presented in the previous section allows us to automatically refine abstractions using the CEGAR paradigm [35]. The pseudo-code of the CEGAR algorithm for

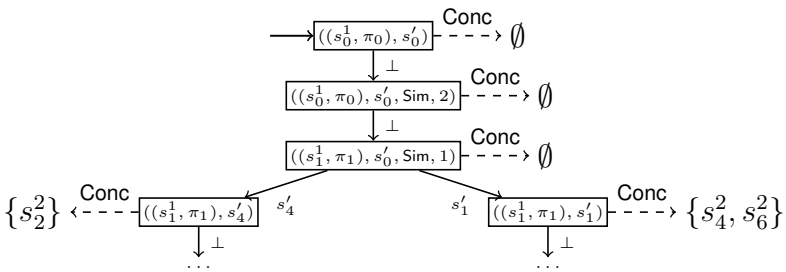

Figure 5.6: Abstract counterexample to the modified combined-simulation game of $(\mathsf{A} \otimes \mathsf{S})$ and G', where $\mathsf{A} = \widehat{G}_1 \parallel Abs^{\Pi}_{\mathcal{A}}(\widehat{G}_2)$ and $\mathsf{S} = \widehat{G}_1 \parallel Abs^{\Pi}_{\mathcal{S}}(\widehat{G}_2)$.

Algorithm 5.5 Assume-guarantee CEGAR for $\leqslant_{\mathcal{C}}$.

Input: Two-player games G_1, G_2, G'.

Output: **Yes** if $G_1 \parallel G_2 \leqslant_{\mathcal{C}} G'$, otherwise **No**

$\Pi \leftarrow$ coarsest partitioning of G_2

loop

$\mathsf{A} \leftarrow G_1 \parallel Abs^{\Pi}_{\mathcal{A}}(G_2)$; $\mathsf{S} \leftarrow G_1 \parallel Abs^{\Pi}_{\mathcal{S}}(G_2)$

$G^{\mathcal{M}} \leftarrow$ modified combined simulation game of $(\mathsf{A} \otimes \mathsf{S})$ and G'

if Player 1 wins in $G^{\mathcal{M}}$ **then** **return** **Yes**

else

Cex $\leftarrow$abstract counterexample in $G^{\mathcal{M}}$

if Feasible(Cex) **then** **return** **No**

else $\Pi \leftarrow$ Refine(Cex, Π)

the assume-guarantee combined simulation is shown in Algorithm 5.5. The algorithm takes G_1, G_2, G' as arguments and answers whether $(G_1 \parallel G_2) \leqslant_{\mathcal{C}} G'$ holds. Initially, the algorithms computes the coarsest partition Π of G_2. Then, it executes the CEGAR loop: in every iteration the algorithm constructs A (resp. S) as the parallel composition of G_1 and the alternating-simulation abstraction (resp. simulation abstraction) of G_2. Let $G^{\mathcal{M}}$ be the modified combined-simulation game of $(\mathsf{A} \otimes \mathsf{S})$ and G'. If Player 1 has a winning strategy in $G^{\mathcal{M}}$ then the algorithm returns **Yes**; otherwise it finds an abstract counterexample Cex in $G^{\mathcal{M}}$. In case the counterexample is feasible, then it corresponds to a concrete counterexample, and the algorithm returns **No**. If Cex is spurious, the algorithm calls a refinement procedure that uses the concretization of Cex to return a partition Π' finer than partition Π. Our technique can be extended to handle multiple

components in a similar way as presented in [110, Section 5].

Refinement procedure Given a partition Π and a spurious counterexample Cex together with its concretization function Conc we describe how to compute the refined partition Π'. Consider a partition $\pi \in \Pi$ and let $\overline{S}_\pi = \{\overline{s}_1, \overline{s}_2, \ldots, \overline{s}_m\}$ denote the states of the abstract counterexample Cex that contain π as its component. Every state $\overline{s}_i$ splits π into at most two sets $\mathsf{Conc}(\overline{s}_i)$ and $\pi \setminus \mathsf{Conc}(\overline{s}_i)$, and let this partition be denoted as T_i. We define a partition $\mathcal{P}_\pi$ as the largest equivalence relation on π that is finer than any of the equivalence relation T_i for all $1 \leq i \leq m$. Formally, $\mathcal{P}_\pi = \{\overline{\pi}_1, \overline{\pi}_2, \ldots, \overline{\pi}_k\}$ is a partition of π such that for all $1 \leq j \leq k$ and $1 \leq i \leq m$ we have $\overline{\pi}_j \subseteq \mathsf{Conc}(\overline{s}_i)$ or $\overline{\pi}_j \subseteq \pi \setminus \mathsf{Conc}(\overline{s}_i)$. The new partition Π' is then defined as the union over $\mathcal{P}_\pi$ for all $\pi \in \Pi$.

Example 5.6. *We continue with our running example. In Example 5.5 we showed that the abstractions of $\widehat{G}_2$ by the coarsest partition Π lead to a spurious counterexample depicted in Figure 5.6. Consider the partition $\pi_1 = \{s_2^2, s_4^2, s_6^2\}$. There are three states in the counterexample that have π_1 as its component and the concretization function assigns to them three subsets of states: $\emptyset, \{s_2^2\}, \{s_4^2, s_6^2\}$. After the refinement partition π_1 is split into two partitions $\pi_1' = \{s_2^2\}$ and $\pi_1'' = \{s_4^2, s_6^2\}$.*

Proposition 5.7. *Given a partition Π and a spurious counterexample Cex, the partition Π' obtained as refinement of Π is finer than Π.*

Sound and completeness of our CEGAR approach Since we consider finite games, the refinement procedure only executes for finitely many steps. In every iteration of the CEGAR algorithm, either the algorithm returns a correct answer (by soundness), or a finer partition is obtained. Thus either we end up with a correct answer, or the trivial partition, and hence the completeness of our approach follows. Thus our CEGAR approach is both sound and complete.

5.8 Experimental Evaluation

We implemented our CEGAR approach for combined simulation in Java, and experimented with our tool on a number of MDPs and two-player games examples. Our algorithms use

explicit representation of MDPs. We use PRISM [62] model checker to specify the examples and generate input files for our tool.

To be compatible with the existing benchmarks (e.g. [110]) in our tool actions are observable instead of atomic propositions. Our algorithms are easily adapted to this setting. We also allow the user to specify silent actions for components, which are not required to be matched by the specification G'.

Improved (modified) combined-simulation game We leverage the fact that MDPs are interpreted as turn-based games to simplify the (modified) combined-simulation game. When comparing two Player-1 states, the last two steps in the alternating-simulation gadget can be omitted, since the players have unique successors given the actions chosen in the first two steps. Similarly, for two probabilistic states, the first two steps in the alternating-simulation gadget can be skipped. We check the (modified) combined-simulation games using the standard attractor algorithm to solve games with safety (as well as reachability) objectives [147; 155].

Improved partition refinement procedure In the implementation we adopt the approach of [105] for refinement. Given a state $\overline{s}$ of the abstract counterexample with partition π as its component, the equivalence relation may split the set $\pi \setminus \mathsf{Conc}(\overline{s})$ into multiple equivalence classes. Intuitively, this ensures that similar-shaped spurious counterexamples do not reappear in the following iterations. This approach is more efficient than the naive one, and also implemented in our tool.

MDP examples with safety specifications We used our tool on all the MDP examples and specifications from [110]. The specifications describe safety properties of the systems.

- **Client-Server protocol** (CS_1 *and* CS_n)

  Model: The example models a Client-Server protocol with mutual exclusion and probabilistic failures in one (CS_1) or all of the clients (CS_n). The model is parametrized by the number of clients.

  Specification: The safety specification is exactly the same as in [110] and characterizes the probabilistic failure model of the clients.

- **Mars Exploration Rover** (MER)

  Model: The example models an arbiter module of NASA's software for Mars Exploration Rovers, which grants shared resources for several users. The number of users is the parameter of the model.

  Specification: The specification is exactly the same as in [110] and imposes a safety requirement on the users' behavior.

- **Sensor networks** (SN)

  Model: The example models a network of sensors that communicate via a bounded buffer with probabilistic behavior in the components. The model is parametrized by the number of sensors.

  Specification: The specification is exactly the same as in [110] and is an abstraction of the observed system behavior.

In addition, we also considered two other classical MDP examples:

- **Leader election protocol** (LE)

  Model: The example is based on a PRISM case study [62] that models the *Leader election protocol* [156], where n agents on a ring randomly pick a number from a pool of K numbers. The agent with the highest number becomes the leader. In case there are multiple agents with the same highest number the election proceeds to the next round. The model is parametrized by the values of n and K, respectively.

  Specification: The specification requires that two leaders cannot be elected at the same time.

- **Peterson's algorithm** (PETP)

  Model: The example is based on Peterson's algorithm [157] for mutual exclusion of n threads, where the execution order is controlled by a randomized scheduler. We extend Peterson's algorithm by giving the threads a non-deterministic choice to restart before entering the critical section. The restart operation succeeds with probability $\frac{1}{2}$ and with probability $\frac{1}{2}$ the thread enters the critical section.

Specification: The specification requires only one process to be in the critical section at any time.

Summary of results for safety specifications For all examples, other than the Client-Server protocol, the assume-guarantee method scales better than the monolithic reasoning; and in all examples our qualitative analysis scales better than the strong simulation approach. Qualitative analysis through combined simulation relies on discrete graph algorithms (attractor computation), while checking strong simulation requires calls to an SMT solver.

Details of experimental results for safety specifications Table 5.1 shows the results for MDP examples we obtained using our assume-guarantee algorithm and the monolithic approach (where the composition is computed explicitly). We also compared our results with the tool presented in [110] that implements both assume-guarantee and monolithic approaches for *strong simulation* [158]. All the results were obtained on a Ubuntu-13.04 64-bit machine running on an Intel Core i5-2540M CPU of 2.60GHz. We imposed a 4.3GB upper bound on Java heap memory and one hour time limit. For $\mathsf{MER}(6)$ and $\mathsf{PETP}(5)$ PRISM cannot parse the input file (probably it runs out of memory).

		AGCS			AGSS			MONCS		MONSS	
Ex.	$\|G_1\|, \|G_2\|, \|G'\|$	$Time$	Mem	$I, \|\Pi\|$	$Time$	Mem	$I, \|\Pi\|$	$Time$	Mem	$Time$	Mem
$\mathsf{CS}_1(5)$	36 / 405 / 16	1.13s	112MB	49 / 85	6.11s	213MB	32 / 33	**0.04s**	**34MB**	0.18s	95MB
$\mathsf{CS}_1(6)$	49 / 1215/ 19	2.52s	220MB	65 / 123	11.41s	243MB	40 / 41	**0.04s**	**51MB**	0.31s	99MB
$\mathsf{CS}_1(7)$	64 / 3645/ 22	5.41s	408MB	84 / 156	31.16s	867MB	56 / 57	**0.05s**	**82MB**	0.77s	113MB
$\mathsf{CS}_n(3)$	125 / 16 / 54	0.65s	102MB	9 / 24	33.43s	258MB	11 / 12	**0.09s**	**35MB**	11.29s	115MB
$\mathsf{CS}_n(4)$	625 / 25 / 189	6.22s	495MB	15 / 42	TO	-	-	**0.4s**	**106MB**	1349.6s	577MB
$\mathsf{CS}_n(5)$	3k / 36 / 648	117.06s	2818MB	24 / 60	TO	-	-	**2.56s**	**345MB**	TO	-
$\mathsf{MER}(3)$	278 / 1728 / 11	**1.42s**	**143MB**	8 / 14	2.74s	189MB	6 / 7	1.96s	228MB	128.1s	548MB
$\mathsf{MER}(4)$	465 / 21k / 14	**4.63s**	**464MB**	13 / 22	10.81s	870MB	10 / 11	11.02s	1204MB	TO	-
$\mathsf{MER}(5)$	700 / 250k / 17	**29.23s**	**1603MB**	20 / 32	67s	2879MB	15 / 16	-	MO	MO	-
$\mathsf{SN}(1)$	43 / 32 / 18	0.13s	38MB	3 / 6	0.28s	88MB	2 / 3	**0.04s**	**29MB**	3.51s	135MB
$\mathsf{SN}(2)$	796 / 32 / 54	0.9s	117MB	3 / 6	66.09s	258MB	2 / 3	**0.38s**	**103MB**	3580.83s	1022MB
$\mathsf{SN}(3)$	7k / 32 / 162	4.99s	**408MB**	3 / 6	TO	-	-	4.99s	612MB	TO	-
$\mathsf{SN}(4)$	52k / 32 / 486	**34.09s**	**2448MB**	3 / 6	TO	-	-	44.47s	3409MB	TO	-
$\mathsf{LE}(3,4)$	4 / 415 / 269	**0.25s**	**70MB**	7 / 16	0.71s	164MB	7 / 8	0.28s	80MB	3.46s	163MB
$\mathsf{LE}(3,5)$	4 / 814 / 513	**0.35s**	**80MB**	7 / 16	Error	-	-	0.81s	157MB	Error	-
$\mathsf{LE}(4,4)$	6 / 5665 / 2561	**1.27**	**128MB**	7 / 19	TO	-	-	12.74s	1186MB	TO	-
$\mathsf{LE}(5,5)$	8 / 29k / 21k	**6.73s**	**517MB**	7 / 21	TO	-	-	TO	-	TO	-
$\mathsf{LE}(6,4)$	10 / 42k / 40k	**8.98s**	**664MB**	7 / 25	TO	-	-	TO	-	TO	-
$\mathsf{LE}(6,5)$	10 / 169k / 56k	**36.38s**	**2372MB**	7 / 25	TO	-	-	TO	-	TO	-
$\mathsf{PETP}(2)$	68 / 3 / 3	0.04s	31MB	0 / 2	0.04s	87MB	0 / 1	0.04s	**30MB**	0.04s	90MB
$\mathsf{PETP}(3)$	4 / 1730 / 4	**0.19s**	**65MB**	6 / 8	0.29s	153MB	3 / 4	0.24s	72MB	1.07s	170MB
$\mathsf{PETP}(4)$	5 / 54k / 5	**1.58s**	**325MB**	8 / 10	3.12s	727MB	4 / 5	7.04s	960MB	31.52s	1741MB

Table 5.1: Results for MDPs examples with safety specifications: AGCS stands for our assume-guarantee combined simulation; AGSS stands for assume-guarantee with strong simulation; MONCS stands for our monolithic combined simulation; and MONSS stands for monolithic strong simulation. The number I denotes the number of CEGAR iterations and $|\Pi|$ the size of the abstraction in the last CEGAR iteration. TO and MO stand for a time-out and memory-out, respectively, and Error means that an error occurred during execution. The memory consumption is obtained using the Unix `time` command.

MDP examples with liveness specifications We have also experimented with MDPs with liveness specifications. We consider the LE and PETP models from the previous safety experiments, as the liveness properties are natural in these models. We also add the additional

Szymanski's mutual-exclusion protocol [159].

		AGCS			MONCS	
Ex.	$\|G_1\|, \|G_2\|, \|G'\|$	$Time$	Mem	$I, \|\Pi\|$	$Time$	Mem
LE$(6,4)$	10 / 42k / 27	**0.82s**	**172MB**	8 / 41	1.66s	275MB
LE$(6,5)$	10 / 169k / 27	**1.73s**	**351MB**	8 / 25	5.05s	808MB
LE$(6,6)$	10 / 521k / 27	**5.01s**	**946MB**	8 / 41	17.3s	2134MB
LE$(7,4)$	12 / 187k / 31	**2.59s**	**447MB**	8 / 52	5.79s	1029MB
LE$(7,5)$	12 / 948k / 31	**12.49s**	**1748MB**	8 / 27	35.49s	3370MB
LE$(7,6)$	12 / 3.5m / 31	**85.61s**	**4303MB**	8 / 27	MO	-
PETP(2)	17 / 184 / 154	0.35s	93MB	3 / 8	**0.18s**	**73MB**
PETP(3)	25 / 10k / 154	**1.23s**	**170MB**	3 / 8	4.79s	593MB
PETP(4)	33 / 864k / 154	**28.61s**	**2187MB**	2 / 8	MO	-
SZYM(2)	24 / 325 / 204	0.5s	108MB	2 / 8	**0.28s**	**99MB**
SZYM(3)	24 / 5010 / 204	**1.1s**	**140MB**	3 / 8	3.34s	407MB
SZYM(4)	24 / 74k / 204	**3.88s**	**343MB**	2 / 8	48.34s	3246MB
SZYM(5)	24 / 1073k / 204	**27.71s**	**2152MB**	2 / 8	MO	-

Table 5.2: Results for MDPs examples with liveness specifications.

- **Leader election protocol** The specification for the leader election protocol LE in addition to the safety property, requires a liveness property that a leader is elected with probability one.

- **Peterson's algorithm** The specification for Peterson's algorithm PETP requires the liveness property that a requesting process is eventually granted access to the critical section with probability one.

- **Szymanski's algorithm** (SZYM)

  Model: The example is based on the Szymanski's algorithm [159] for mutual exclusion of n threads with a randomized scheduler. Threads were additionally extended by a non-deterministic choice to wait, rather than only request access to the critical section.

  Specification: As for PETP, the specification requires that a requesting process eventually enters the critical section with probability one.

Table 5.2 shows the experimental results for systems with liveness specification. We did not compare our results with the tool prototype from [110], which checks strong simulation, since the tool was tailored to safety specifications only. For all the examples, the assume-guarantee method scales better than the monolithic method both in time and memory.

Two-player games examples We also experimented with our tool on several examples of games, where one of the players controls the choices of the system and the other player represents the environment.

- **Error-correcting device** (EC)

  Model: The example is based on [160] and models an error-correcting device that sends and receives data blocks over a communication channel. Notation $\mathsf{EC}(n, k, d)$ means that a data block consists of n bits and it encodes k bits of data; value d is the minimum Hamming distance between two distinct blocks. In the first component Player 2 chooses a message to be sent over the channel and is allowed to flip some bits in the block during the transmission. The second component restricts the number of bits that Player 2 can flip.

  Specification: The specification requires that every message is correctly decoded.

- **Peterson's algorithm** (PETG)

  Model: The model is similar to the Peterson's algorithm [157] example for MDPs, with the following differences: (a) the system may choose to restart instead of entering the critical section; (b) instead of a randomized scheduler we consider an adversarial scheduler.

  Specification: The specification requires only one process to be in the critical section at any time.

- **Virus attack** (VIR1)

  Model: The example models a virus that attacks a computer system with n nodes (based on case study from PRISM [62]). Player 1 represents the virus and is trying to infect as many nodes of the network as possible. Player 2 represents the system and may

recover an infected node to an uninfected state. VIR2 is a modified version of VIR1 with two special critical nodes in the network. Whenever both of the nodes are infected, the virus can overtake the system.

Specification: The specification for VIR1 requires that the virus has a strategy to avoid being completely erased, i.e., maintain at least one infected node in the network. The specification for VIR2 is like for VIR1, i.e., the virus can play such that at least one node in the network remains infected, but it additionally requires that even if the system cooperates with the virus, the system is designed in a way that the special nodes will never be infected at the same time.

The results for two-player game examples are shown in Table 5.3. Along with AGCS and MONCS for assume-guarantee and monolithic combined simulation, we also consider AGAS and MONAS for assume-guarantee and monolithic alternating simulation, as for properties in 1-ATL it suffices to consider only alternating simulation. For all the examples, the assume-guarantee algorithms scale better than the monolithic ones. Combined simulation is finer than alternating simulation and therefore combined simulation may require more CEGAR iterations.

		AGCS			MONCS		AGAS			MONAS	
Ex.	$\|G_1\|, \|G_2\|, \|G'\|$	$Time$	Mem	$I, \|\Pi\|$	$Time$	Mem	$Time$	Mem	$I, \|\Pi\|$	$Time$	Mem
EC$(32, 6, 16)$	71k / 193 / 129	3.55s	446MB	1 / 7	**1.15s**	281MB	2.34s	391MB	0 / 2	1.03s	**251MB**
EC$(64, 7, 16)$	549k / 385 / 257	70.5s	3704MB	1 / 131	9.07s	1725MB	16.79s	1812MB	0 / 2	**4.83s**	**1467MB**
EC$(64, 8, 16)$	1.1m / 769 / 513	-	MO	-	-	MO	**52.63s**	**3619MB**	0 / 2	-	MO
EC$(64, 8, 32)$	1.1m / 1025 / 513	-	MO	-	-	MO	**54.08s**	**3665MB**	0 / 2	-	MO
PETG(2)	3 / 52 / 3	0.08s	35MB	4 / 6	0.03s	30MB	0.07s	35MB	4 / 6	0.03s	**29MB**
PETG(3)	4 / 1514 / 4	**0.2s**	63MB	6 / 8	0.25s	74MB	0.22s	**62MB**	6 / 8	0.21s	64MB
PETG(4)	5 / 49k / 5	1.75s	316MB	8 / 10	8.16s	1080MB	**1.6s**	**311MB**	8 / 10	6.94s	939MB
VIR1(12)	14 / 4097 / 1	0.91s	159MB	15 / 30	1.69s	255MB	**0.35s**	**114MB**	2 / 4	1.53s	215MB
VIR1(13)	15 / 8193 / 1	1.47s	197MB	16 / 32	4.36s	601MB	**0.6s**	**178MB**	2 / 4	2.8s	402MB
VIR1(14)	16 / 16k / 1	3.09s	326MB	17 / 34	8.22s	992MB	**0.75s**	**241MB**	2 / 4	6.49s	816MB
VIR1(15)	17 / 32k / 1	4.47s	643MB	18 / 36	15.13s	2047MB	**1.05s**	**490MB**	2 / 4	9.67s	1361MB
VIR1(16)	18 / 65k / 1	8.65s	1015MB	19 / 38	41.28s	3785MB	**1.37s**	**839MB**	2 / 4	23.71s	2591MB
VIR1(17)	19 / 131k / 1	18.68s	1803MB	20 / 40	-	MO	**2.12s**	**1653MB**	2 / 4	62.24s	4309MB
VIR1(18)	20 / 262k / 1	38.68s	3079MB	21 / 42	-	MO	**3.35s**	**2878MB**	2 / 4	-	MO
VIR2(12)	13 / 4096 / 1	1.02s	151MB	19 / 34	0.81	154MB	0.68s	**122MB**	9 / 14	**0.57s**	133MB
VIR2(13)	14 / 8192 / 1	1.48s	190MB	20 . 36	1.13s	216MB	1.01s	**183MB**	9 / 14	1.01s	208MB
VIR2(14)	15 / 16k / 1	2.9s	315MB	21 / 38	2.33s	389MB	**1.94s**	**311MB**	9 / 14	2.09s	388MB
VIR2(15)	16 / 32k / 1	5s	631MB	22 / 40	6.29s	964MB	**2.12s**	**489MB**	9 / 14	4.69s	757MB
VIR2(16)	17 / 65k / 1	9.82s	949MB	23 / 42	7.55s	1468MB	**3.96s**	**897MB**	9 / 14	6.09s	1315MB
VIR2(17)	18 / 131k / 1	23.33s	1815MB	24 / 44	23.54s	3012MB	**8.16s**	**1676MB**	9 / 14	15.36s	2542MB
VIR2(18)	19 / 262k / 1	45.89s	3049MB	25 / 46	55.28s	4288MB	**20.3s**	**2875MB**	9 / 14	28.79s	3755MB

Table 5.3: Results for two-player games examples.

6 Array Folds Logic

6.1 Introduction

Arrays and lists (or, more generally, sequences) are fundamental data structures both for imperative and functional programs: hardly any real-life program can work without processing sequentially-ordered data. Testing and verification of array- and list-manipulating programs is thus a task of crucial importance. Almost any non-trivial property about these data structures requires some sort of universal quantification; unfortunately, the full first-order theories of arrays and lists are undecidable. This has motivated researchers to investigate fragments with restricted quantifier prefixes, and has given rise to numerous logics that can describe interesting properties of sequences, such as partitioning or sortedness. These logics have efficient decision procedures and have been successfully applied to verify some important aspects of programs working with arrays and lists: for example, the correctness of sorting algorithms.

However, an important class of properties, namely, *counting* over arrays, has eluded researchers' attention so far. In addition to the examples from the abstract, this includes statements such as "the histogram of the input data satisfies the given distribution," or "the packet adheres to the requirements of the given type-length-value (TLV) encoding (e.g., of the IPv6 options)." Such properties, though crucial for many applications, cannot be expressed in decidable fragments of the first-order theory of arrays, nor in the decidable extensions of the theory of concatenation.

In this chapter we present *array folds logic* (AFL), which is an extension of the quantifier-free theory of integer arrays. But instead of introducing quantifiers, we introduce counting in the form of *fold* terms. Folding is a well-known concept in functional languages: as the name suggests, it folds some function over an array, i.e., applies it to every element of the array in sequence, while

preserving the intermediate result.

```
min = max = a[0];
j = k = 0;
for(i=0;i<size(a);i++) {
  if(a[i]<min) { min=a[i]; j=1; }
  if(a[i]==min) j++;
}
for(i=0;i<size(a);i++) {
  if(a[i]>max) { max=a[i]; k=1; }
  if(a[i]==max) k++;
}
assert(j==k);
```

(a) C language.

$$\begin{aligned}
&\exists\, min, max, i_1, i_2, j, k \ : \\
&0 \leq i_1 < |a| \ \wedge \ 0 \leq i_2 < |a| \ \wedge \\
&a[i_1] = min \ \wedge \ a[i_2] = max \ \wedge \\
&\forall i : (a[i] \geq min) \ \wedge \\
&\forall i : (a[i] \leq max) \ \wedge \\
&j = \big|\{i \mid a[i] = min\}\big| \ \wedge \\
&k = \big|\{i \mid a[i] = max\}\big| \ \wedge \\
&j = k
\end{aligned}$$

(b) Quantified arrays + cardinality.

$$\begin{gathered}
0 \leq i_1 < |a| \ \wedge \ 0 \leq i_2 < |a| \ \wedge \ a[i_1] = min \ \wedge \ a[i_2] = max \ \wedge \\
\mathit{fold}_a \begin{pmatrix} 0 \\ 0 \end{pmatrix} \begin{pmatrix} \mathbf{e}{=}min \Rightarrow \mathbf{c}_1{++} \\ \mathbf{e}{>}min \Rightarrow \mathit{skip} \end{pmatrix} = \begin{pmatrix} |a| \\ j \end{pmatrix} \ \wedge \ \mathit{fold}_a \begin{pmatrix} 0 \\ 0 \end{pmatrix} \begin{pmatrix} \mathbf{e}{=}max \Rightarrow \mathbf{c}_1{++} \\ \mathbf{e}{<}max \Rightarrow \mathit{skip} \end{pmatrix} = \begin{pmatrix} |a| \\ k \end{pmatrix} \ \wedge \ j = k
\end{gathered}$$

(c) Array folds logic.

Figure 6.1: A toy array problem.

To illustrate the kind of problems we are dealing with, consider the following toy example: given an array, accept it if the number of minimum elements in the array is the same as the number of maximum elements in the array. E.g., the array $[1, 2, 7, 4, 1, 3, 7, 5]$ is accepted (because there are two 1's and two 7's), while the array $[1, 2, 7, 4, 1, 3, 6, 5]$ is rejected (because there is only one 7).

Written in a programming language like C, the problem can be solved by the piece of code shown in Figure 6.1a, but such *explicit* solution cannot express verification conditions for *symbolic* verification and testing. We can use the quantified theory of arrays mixed with assertions about cardinality of sets, as in Figure 6.1b. Unfortunately, such a combination is undecidable (by a reduction from Hilbert's Tenth Problem: replace *fold*s with cardinalities in the

proof of Theorem 6.2).

The solution we propose is shown in Figure 6.1c: in the example formula, the first $fold$ applies a function to array a. The vector in the first parentheses gives initial values for the array index and counter $\mathbf{c}_1$; the function is folded over the array starting from the initial index. Index variable $\mathbf{i}$ is implicit, and it is incremented at each iteration. The function itself is given in the second parentheses, and has two branches. The first branch *counts* the number of positions with elements equal to min in counter $\mathbf{c}_1$. The second branch $skip$s when the current array element $\mathbf{e}$ is greater than the (guessed, existentially quantified) variable min. When $\mathbf{e} < min$, the implicit $break$ statement is executed, and the $fold$ terminates prematurely. The result of the $fold$ is compared to the vector which asserts that the final value of the array index equals to the array size $|a|$ (which means no $break$ was executed), and the final value of $\mathbf{c}_1$ equals to j. The positions where elements are equal to max, are counted in the second $fold$, and the equality between these two counts is asserted. The ability to count over arrays with unbounded elements is a unique feature of array folds logic.

Contributions Our contributions are as follows:

- We define a new logic, called AFL, that can express interesting and non-trivial properties of counting over arrays, which are orthogonal to the properties expressible by other logics. Additionally, AFL can concisely summarize loops with internal branching that traverse arrays and perform counting, enabling verification and symbolic testing of programs with such loops.

- We show that the satisfiability problem for AFL is $\mathbf{PSPACE}$-complete, and with a natural restriction the complexity decreases to $\mathbf{NP}$. We provide a decision procedure for AFL, which works by a reduction to the emptiness of (symbolic) reversal-bounded counter machines, which in turn reduces to the satisfiability of existential Presburger formulas. We show that adding either universal quantifiers or concatenation leads to undecidability.

- We implemented a tool AFOLDER [161] that can discharge proof obligations in AFL, and we demonstrate on real-life examples that our decision procedure can solve a broad range of problems in symbolic testing and program verification.

6.2 Related Work

Our logic is related to the quantified fragments of the theory of arrays such as [24][37][38][39]. These logics allow restricted quantifier prefixes, and their decision procedures work by rewriting to the (parametric) theories of array indices and elements (Presburger arithmetic being the most common case) [24][39], or by reduction to flat counter automata with difference bound constraints [37][38]. An interesting alternative is provided in [162], where the quantification is arbitrary, but array elements must be bounded by a constant given a priori; the decision procedure works by a reduction to WS1S. A separate line of work is presented by the theory combination frameworks of [163][164], where the quantifier-free theory of arrays is extended by $injective$ predicate and $domain$ function [163], or with map and $constant\text{-}value$ combinators [164]. The theory of concatenation and its extensions [28][165][166] are also related; their decision procedures work by reduction to Makanin's algorithm for solving word equations [167]. AFL can express some properties that are also expressible in these logics, such as *boundedness*, *partitioning*, or *periodicity*; other properties, such as *sortedness*, are not expressible in AFL. The counting properties that constitute the core of AFL are not expressible in any of the above logics. We compare the expressive power of AFL and other logics in Section 6.3.3.

There are numerous works on loop acceleration and summarization [168][169][170], also in the context of verification and symbolic testing [171][172][173][174] and array-manipulating programs [175][176][177]. Our logic allows one to summarize loops with internal branching and counting, which are outside of the scope of these works.

The decision procedure for AFL is based on the decidability results for emptiness of reversal-bounded counter machines [178][179][180], on the encoding of this problem into Presburger arithmetic [181], and on the computation of Parikh images for NFAs [182]. In Section 6.6 we extend the encoding procedure to symbolic counter machines, and present some substantial improvements that make it efficient for solving practical AFL problems.

6.3 Syntax and Semantics

We assume familiarity with the standard syntax and terminology of many-sorted first-order logics. We use vector notation: $\mathbf{v} = (v_1, \ldots, v_n)$ denotes an ordered sequence of terms. The i-th

element of $\mathbf{v}$ is denoted by $\mathbf{v}(i) = v_i$. For two vectors $\mathbf{u}$ and $\mathbf{v}$, we write their concatenation as $\mathbf{uv}$.

In this chapter we consider the domains of arrays, array indices, and array elements to be $\mathbb{A} = \mathbb{Z}^*$, $\mathbb{N} = \{0, 1, \dots\}$, and $\mathbb{Z} = \{\dots, -1, 0, 1, \dots\}$, respectively. Presburger arithmetic has the signature $\Sigma_{\mathbb{Z}} = \{0, 1, +, <\}$; we use it for array indices and elements, as well as other arithmetic assertions, possibly with embedded array terms. We write true and false to denote a valid and an unsatisfiable Presburger formula, respectively.

The theory of integer-indexed arrays extends Presburger arithmetic with functions $read$, and $write$, and has the signature $\Sigma_A = \Sigma_{\mathbb{Z}} \cup \{\cdot[\cdot], \cdot\{\cdot \leftarrow \cdot\}\}$. The $read$ function $a[i]$ returns the i-th element of array a, and the $write$ function $a\{i \leftarrow x\}$ returns array a where the i-th element is replaced by x. These functions should satisfy the *read*-over-*write* axioms as described by McCarthy [183]; the decision procedure for the quantifier-free array theory is presented in [184].

6.3.1 Syntax

*Array folds logic (*AFL*)* extends the quantifier-free theory of integer arrays with the ability to perform counting. The extension works by incorporating $fold$ terms into arithmetic expressions; such a term folds some function over the array by applying it to each array element consecutively.

AFL contains the following sorts: array sort ASort, integer sort ISort, Boolean sort BSort, and two enumerable sets of sorts for integer vectors VSort^m and functional constants $\mathsf{FSort}^m = \mathsf{VSort}^m \times \mathsf{ISort} \rightarrow \mathsf{VSort}^m$, for each $m \in \mathbb{N}$, $m > 0$. The syntax of the AFL terms is shown in Table 6.1; a and b denote array variables, x denotes an integer variable, n and m denote integer constants.

Array terms A of the sort ASort are represented either by an array variable a, or by the $write$ term $a\{T \leftarrow T\}$.

Integer terms T of the sort ISort can be integer constants $n \in \mathbb{Z}$, integer variables x, integer addition, $read$ term $A[T]$ for the index represented as an integer term, or the term $|a|$, which represents the length of array a.

Boolean terms B of the sort BSort are formed by the standard Presburger and Boolean operators, and equality between vectors of the sort VSort^m.

Vector terms V^m of the sort VSort^m are either a list of m integer terms, or a $fold$ term. The

former is written as a vertical list in parentheses; they can be omitted when $m = 1$. The latter, written as $fold_a\ v\ f$, represents the result of the transformation of an input vector v of the sort VSort^m by folding a functional constant f of the sort FSort^m over an array a. The first element of v specifies an initial value of the array index; the remaining elements give initial values for the counters that can be used inside f. The resulting vector after the transformation gives the final values for the array index and the counters.

Functional constants (when no confusion can arise, we call them *functions*) F^m of the sort FSort^m can only be a parenthesized list of branches (guarded commands); the length of the list is unrelated to m. A function f of the sort FSort^m can refer to the following implicitly declared variables: $\mathbf{e}$ for the currently inspected array element; $\mathbf{i}$ for the current array index; $\mathbf{c}_1, \ldots, \mathbf{c}_{m-1}$ for the counters; $\mathbf{s}$ for the state (control flow) variable. All other variables that occur inside f are considered as free variables of the sort ISort.

Guards are conjunctions of *atomic guards*, which can compare array elements, indices, and counters to integer terms; the state variable can only be compared to integer constants. *Updates* are lists of *atomic updates*; they can increment or decrease counters by a constant, assign a constant to the state variable, $skip$, i.e. perform no updates, or execute a $break$ statement, which terminates the $fold$ at the current position. Counter or state updates define a function $\mathbb{Z} \to \mathbb{Z}$. Guards and updates translate into logical formulas that either constrain the current variable values, or relate the current and the next-state (primed) variable values in the obvious way; we denote this translation by Φ. E.g., the update $upd \equiv (\mathbf{c}_1 \mathrel{+}= n)$ defines the formula $\Phi(upd) \equiv (\mathbf{c}_1' = \mathbf{c}_1 + n)$.

The size $|\phi|$ of an AFL formula is the length of the binary encoding of ϕ. We require that guards of all branches are mutually exclusive. There is an implicit "catch-all" branch with the $break$ statement, whose guard evaluates to true exactly when guards of all other branches evaluate to false. We also require that each branch contains at most one update for each implicit variable.

We restrict the control flow in functions, which is defined by state variable $\mathbf{s}$. Notice that $\mathbf{s}$ is syntactically finite state. Thus, given a set of function branches Br, we define an edge-labeled control flow graph $G = \langle Q, E, \gamma \rangle$, where:

- states $Q = \{0\} \cup \{\, n \mid \mathbf{s} \leftarrow n \in Br \,\}$;

$$
\begin{array}{lll}
A & ::= a \mid a\{T \leftarrow T\} & \\
T & ::= n \mid x \mid T + T \mid A[T] \mid |a| & \\
B & ::= T = T \mid T < T \mid \neg B \mid B \wedge B \mid V^m = V^m & \\
V^m & ::= \begin{pmatrix} T \\ \cdots \\ T \end{pmatrix} \mid \mathit{fold}_a\, V^m\, F^m & \\
F^m & ::= \begin{pmatrix} \mathit{grd} \Rightarrow \mathit{upd} \\ \cdots \\ \mathit{grd} \Rightarrow \mathit{upd} \end{pmatrix} & \\
\mathit{grd} & ::= \mathbf{e} \approx T \mid \mathbf{i} \approx T \mid \mathbf{c}_m \approx T \mid \mathbf{s} \approx n \mid \mathit{grd} \wedge \mathit{grd} & (\approx \in \{>, <, =, \neq\}) \\
\mathit{upd} & ::= \mathbf{c}_m \mathrel{+}= n \mid \mathbf{s} \leftarrow n \mid \mathit{skip} \mid \mathit{break} \mid \mathit{upd}\ ;\ \mathit{upd} &
\end{array}
$$

Table 6.1: Syntax of AFL.

- edges $E = \bigcup_{\mathit{grd} \Rightarrow \mathit{upd} \in Br} \{\ (s_1, s_2) \mid \mathit{grd}[\mathbf{s}/s_1] \text{ is sat } \wedge s_2 = \mathit{ite}(\mathbf{s} \leftarrow n \in \mathit{upd}, n, s_1)\ \}$;
- γ is the labeling of edges with the set of formulas $\Phi(\mathit{grd})$ and $\Phi(\mathit{upd})$ for each guard or update which occurs in the same branch.

We require that edges in the strongly-connected components of G are labeled with counter updates that are, for each counter, all non-decreasing, or all non-increasing. Thus, G is a DAG of strongly connected components (SCC), where counters within each SCC behave in a monotonic way. We use this restriction to derive from f a reversal-bounded counter machine (see Definition 6.2).

The presented syntax is minimal and can be extended with convenience functions and predicates such as $\{-, n\cdot, , \leq, \geq, \vee, ++, -, -{=}n\}$ in the usual way. We allow to use $*$ to denote the absence of constraints: this is useful for vector notation. We replace each $*$ in the formula with a unique unconstrained variable.

6.3.2 Semantics

For a given AFL formula ϕ, we denote the sets of free variables of ϕ of the sort ASort and ISort by Var_A and Var_I, respectively. All free variables are implicitly existentially quantified. For

1.	$[V_1^m = V_2^m]^\sigma$	$\equiv \forall i:\ 1 \leq i \leq m \implies [V_1^m]^\sigma(i) = [V_2^m]^\sigma(i)$
2.	$[fold_a\ v\ f]^\sigma$	$\equiv [fold_a\ v\ f]^{\sigma,\kappa}$, where $\kappa(FV^m) = \begin{pmatrix} [v]^\sigma \\ 0 \\ [a]^\sigma \end{pmatrix}$
3.	$[fold_a\ v\ f]^{\sigma,\kappa}$	$\equiv$ if $([\mathbf{i}]^\kappa < 0)$ or $([\mathbf{i}]^\kappa \geq \mid [a]^\sigma \mid)$ or $(false \in [f]^{\sigma,\kappa})$ then v', where $\begin{pmatrix} v' \\ \mathbf{s}' \\ \alpha \end{pmatrix} = \kappa(FV^m)$
		else $[fold_a\ v\ f]^{\sigma,\kappa'}$, where $\kappa'(FV^m) = \begin{pmatrix} v' \\ \mathbf{s}' \\ \alpha \end{pmatrix} = [f]^{\sigma,\kappa}\left(\kappa(FV^m)\right)$
4.	$[f]^{\sigma,\kappa}\begin{pmatrix} v_1 \\ \dots \\ v_m \end{pmatrix}$	$\equiv \begin{pmatrix} v_1' \\ \dots \\ v_m' \end{pmatrix}$, where $v_j' \equiv$ if $upd(v_j) \in [f]^{\sigma,\kappa}$ then $upd(v_j)$, else v_j
5.	$\left[\begin{pmatrix} grd_1 \Rightarrow upd_1 \\ \dots \\ grd_m \Rightarrow upd_m \end{pmatrix}\right]^{\sigma,\kappa}$	$\equiv \{\mathbf{i}' = \mathbf{i}+1\} \cup [grd_1 \Rightarrow upd_1]^{\sigma,\kappa} \cup \ldots \cup [grd_m \Rightarrow upd_m]^{\sigma,\kappa}$
6.	$[grd \Rightarrow upd]^{\sigma,\kappa}$	$\equiv$ if $[grd]^{\sigma,\kappa} = true$ then $[upd]^{\sigma,\kappa}$ else $\emptyset$
7.	$[\mathbf{e} \approx t]^{\sigma,\kappa}$	$\equiv [\alpha]^\kappa([\mathbf{i}]^\kappa) \approx [t]^\sigma$ (similarly for $\mathbf{i} \approx T,\ \mathbf{c}_m \approx T,\ \mathbf{s} \approx n$)
8.	$[grd_1 \wedge grd_2]^{\sigma,\kappa}$	$\equiv [grd_1]^{\sigma,\kappa} \wedge [grd_2]^{\sigma,\kappa}$
9.	$[upd_1; upd_2]^{\sigma,\kappa}$	$\equiv [upd_1]^{\sigma,\kappa} \cup [upd_2]^{\sigma,\kappa}$
10.	$[\mathbf{c}_m \mathrel{+}= n]^{\sigma,\kappa}$	$\equiv \{\mathbf{c}_m' = \mathbf{c}_m + n\}$
11.	$[\mathbf{s} \leftarrow n]^{\sigma,\kappa}$	$\equiv \{\mathbf{s}' = n\}$
12.	$[skip]^{\sigma,\kappa}$	$\equiv \emptyset$
13.	$[break]^{\sigma,\kappa}$	$\equiv \{false\}$

Table 6.2: Semantics of AFL

functions of the sort FSort^m, we denote by FV^m the set of their implicit variables and a special variable α that denotes the value of the array being folded, i.e. $FV^m = \{\mathbf{i}, \mathbf{c}_1, \ldots, \mathbf{c}_{m-1}, \mathbf{s}, \alpha\}$.

The treatment of array writes and reads in the quantifier-free array theory is standard [184], and we do not elaborate on it here. Array equalities partition the set of array variables into equivalence classes; all other constraints are then translated into constraints over a representative of the corresponding equivalence class.

An *interpretation* for AFL is a tuple $\sigma = \langle \lambda, \mu \rangle$, where $\lambda : Var_I \to \mathbb{Z}$ assigns each integer variable an integer, and $\mu : Var_A \to \mathbb{Z}^*$ assigns each array variable a finite sequence of integers.

The semantics of an AFL term t under the given interpretation σ is defined by the *evaluation*

$[t]^{\sigma}$. Terms that constitute functions are evaluated in the additional *context* κ. For a function f of the sort FSort^m, $\kappa : FV^m \to \mathbb{Z} \cup \mathbb{Z}^*$ maps internal variables of f to integers, and the special variable α to the value of the array being folded. The evaluation of Presburger, Boolean, and array terms is standard; the remaining ones are shown in Table 6.2. We give some explanations here (the remaining semantic rules are self-explanatory):

1. Vector equality resolves to equality between components.

2. A $fold$ term evaluates in the initial context that is defined by the given initial vector of counters v, assigns 0 to the state variable $\mathbf{s}$, and map the variable α to the value of the folded array.

3. A contextual $fold$ term checks whether the array index is out of bounds, or a $break$ statement is executed in the current context (this is the only way for $[f]^{\sigma,\kappa}$ to contain false). If yes, $fold$ terminates, and returns the current values of the array index and counters. Otherwise $fold$ continues with the updated vector and context.

4. If an update $upd(v_j)$ for some variable v_j is present in the function evaluation, then it is applied. Otherwise, the old variable value is preserved.

5. An evaluation of a function, represented by a list of branches, is a union of updates from its branch evaluations. Index $\mathbf{i}$ is always incremented by 1.

6. A guarded command evaluates to its update if its guard evaluates to $true$.

7. A comparison over an internal variable evaluates it in the context κ, and the comparison term is evaluated in the interpretation σ.

6.3.3 Expressive Power

Here we give some example properties that are expressible in AFL, and compare its expressive power to other decidable array logics.

1. **Boundedness**. All elements of array a belong to the interval $[l, u]$.

$$fold_a\Big(0\Big)\Big(_{l \leq \mathbf{e} \leq u \,\Rightarrow\, skip}\Big) = |a|$$

2. **Partitioning**. Array a is partitioned if there is a position p such that all elements before p are smaller or equal than all elements at or after p.

$$fold_a\Big(0\Big)\begin{pmatrix}\mathbf{i}{<}p \wedge \mathbf{e}{\leq}a[p] \Rightarrow skip\\ \mathbf{i}{\geq}p \wedge \mathbf{e}{\geq}a[p] \Rightarrow skip\end{pmatrix} = |a|$$

3. **Periodicity**. Array a is of the form $(01)^*$:

$$fold_a\Big(0\Big)\begin{pmatrix}\mathbf{s}{=}0 \wedge \mathbf{e}{=}0 \Rightarrow \mathbf{s}{\leftarrow}1\\ \mathbf{s}{=}1 \wedge \mathbf{e}{=}1 \Rightarrow \mathbf{s}{\leftarrow}0\end{pmatrix} = |a|$$

4. **Pumping**. Array a is of the form 0^n1^n (a canonical non-regular language; $0^n1^n2^n$, a non-context-free language, is equally expressible):

$$fold_a\begin{pmatrix}0\\0\\0\end{pmatrix}\begin{pmatrix}\mathbf{s}{=}0 \wedge \mathbf{e}{=}0 \Rightarrow \mathbf{c}_1{++}\\ \mathbf{s}{=}0 \wedge \mathbf{e}{=}1 \Rightarrow \mathbf{c}_2{++} \wedge \mathbf{s}{\leftarrow}1\\ \mathbf{s}{=}1 \wedge \mathbf{e}{=}1 \Rightarrow \mathbf{c}_2{++}\end{pmatrix} = \begin{pmatrix}|a|\\n\\n\end{pmatrix}$$

5. **Equal Count**. Arrays a and b have equal number of elements greater than l:

$$\begin{pmatrix}|a|\\n\end{pmatrix} = fold_a\begin{pmatrix}0\\0\end{pmatrix}\begin{pmatrix}\mathbf{e}{>}l \Rightarrow \mathbf{c}_1{++}\\ \mathbf{e}{\leq}l \Rightarrow skip\end{pmatrix} \;\wedge\; \begin{pmatrix}|b|\\n\end{pmatrix} = fold_b\begin{pmatrix}0\\0\end{pmatrix}\begin{pmatrix}\mathbf{e}{>}l \Rightarrow \mathbf{c}_1{++}\\ \mathbf{e}{\leq}l \Rightarrow skip\end{pmatrix}$$

6. **Histogram**. The histogram of the input data in array a satisfies the distribution $H\big(\{i|a[i] < 10\}\big) \geq 2H\big(\{i \mid a[i] \geq 10\}\big)$:

$$fold_a\begin{pmatrix}0\\0\end{pmatrix}\begin{pmatrix}\mathbf{e}{<}10 \Rightarrow \mathbf{c}_1{++}\\ \mathbf{e}{\geq}10 \Rightarrow skip\end{pmatrix} = \begin{pmatrix}|a|\\h_1\end{pmatrix} \;\wedge\; fold_a\begin{pmatrix}0\\0\end{pmatrix}\begin{pmatrix}\mathbf{e}{\geq}10 \Rightarrow \mathbf{c}_1{++}\\ \mathbf{e}{<}10 \Rightarrow skip\end{pmatrix} = \begin{pmatrix}|a|\\h_2\end{pmatrix} \;\wedge\; h_1 \geq 2h_2$$

7. **Length of Format Fields**. The array contains two variable-length fields. The first two elements of the array define the length of each field; they are followed by the fields themselves, separated by 0:

$$len_1 = a[0] \;\wedge\; len_2 = a[1] \;\wedge\; fold_a\begin{pmatrix}2\\0\\0\end{pmatrix}\begin{pmatrix}\mathbf{s}{=}0 \wedge \mathbf{e}{\neq}0 \Rightarrow \mathbf{c}_1{++}\\ \mathbf{s}{=}0 \wedge \mathbf{e}{=}0 \Rightarrow \mathbf{s}{\leftarrow}1\\ \mathbf{s}{=}1 \wedge \mathbf{e}{\neq}0 \Rightarrow \mathbf{c}_2{++}\end{pmatrix} = \begin{pmatrix}|a|\\len_1\\len_2\end{pmatrix}$$

Comparison with other logics. Most decidable array logics can specify universal properties over a *single* index variable like (1) above; AFL uses *fold*s to express such universal quantification. Properties that require universal quantification over *several* index variables, like sortedness, are inexpressible in AFL (it can simulate some of such properties, like partitioning (2), using a combination of *fold*s with existential guessing). Periodic facts like (3) are inexpressible in [24], but AFL as well as [37][165] can express it. Counting properties such as (4)–(7), which constitute the core of AFL, are not expressible in other decidable logics over arrays and sequences.

6.4 Motivating Example

As a motivating example to illustrate applications of our logic, we consider a parser for the Markdown language as implemented in the Redcarpet project, hosted on GitHub [185]. Redcarpet is a popular implementation of the language, used by many other projects, in particular by the GitHub itself. Figure 6.2 shows the excerpt from the function `parse_table_header`, which can be found in the file `markdown.c`.

The function considered in the example parses the header of a table in the Markdown format. The first line of the header specifies column titles; they are separated by pipe symbols ('`|`'); the first pipe is optional. Thus, the number of pipes defines the number of columns in the table. The second line describes the alignment for each column, and should contain the same number of columns; in between each pair of pipes there should be at least three dash ('`-`') or colon ('`:`') symbols. A colon on the left or on the right side of the dashes defines left or right alignment; colons on both sides mean centered text. Thus, the two lines "`|One|Two|Three|`" and "`|:--|:--:|--:|`" specify three columns which are left-, center-, and right-aligned. Replacing the second line with either "`|:-|:--:|--:|`" or "`|:--|:--:|`" would result in the ill-formed input: the former doesn't contain enough dashes in the first column, while the latter doesn't specify the format for the last column.

Suppose, we are interested in the symbolic testing of the parser implementation; in particular, we want to cover all branches in the code for a reasonably long input. For that we postulate that the first input line contains at least n columns (we add the condition `assert(col>=n)` after line 19).

Now, consider the last conditional statement at line 19. The `if` branch is satisfied by an empty second input line; and indeed, concolic testers such as CREST can easily cover it. The `else` branch, however, poses serious problems. In order to cover it, a well-formed input that respects all constraints should be generated; in particular the smallest length of such input, e.g., for n equal to 3, is 17. The huge number of combinations to test exceeds the capabilities of the otherwise very efficient concolic tester: for $n = 2$ CREST needs 800 seconds to generate a test, and for $n = 3$ it is not able to finish within 3 hours.

Let us now examine the encoding of the implementation semantics in array folds logic. The AFL assertions are shown in Figure 6.2 intertwined with the source code: they encode the

```
 1: static size_t parse_table_header(uint8_t *a, size_t size, ...)
 2:   size_t i=0, pipes=0;
```

$$\left\{ i_0 = 0 \wedge p_0 = 0 \right\}$$

```
 3:   while (i < size && a[i] != '\n')
 4:     if (a[i++] == '|') pipes++;
```

$$\left\{ \binom{i_1}{p_1} = \mathit{fold}_a\binom{i_0}{p_0}\left(\begin{smallmatrix}\mathbf{e}=P \Rightarrow \mathbf{c}_1{++} \\ \mathbf{e}\neq P \wedge \mathbf{e}\neq N \Rightarrow \mathit{skip}\end{smallmatrix}\right) \right\}$$

```
 5:   if (a[0] == '|') pipes--;
```

$$\left\{ \binom{*}{p_2} = \mathit{fold}_a\binom{0}{p_1}\left(\mathbf{i} = 0 \wedge \mathbf{e} = P \Rightarrow \mathbf{c}_1{-}\right) \right\}$$

```
 6:  i++;
 7:  if (i < size && a[i] == '|') i++;
```

$$\left\{ i_2 = i_1 + 1 \wedge i_3 = \mathit{fold}_a\left(i_2\right)\left(\mathbf{i} = i_2 \wedge \mathbf{e} = P \Rightarrow \mathit{skip}\right) \right\}$$

```
 8:   end = i;
 9:   while (end < size && a[end] != '\n') end++;
```

$$\left\{ e_0 = i_3 \wedge e1 = \mathit{fold}_a\left(e_0\right)\left(\mathbf{e} \neq N \Rightarrow \mathit{skip}\right) \right\}$$

```
10:  for (col = 0; col<pipes && i<end; ++col) {
11:    size_t dashes = 0;
```

$$\left\{ c_0 = 0 \wedge c_0 < p_2 \wedge i_3 < e_1 \wedge d_0 = 0 \right\}$$

```
12:    if (a[i] == ':') { i++; dashes++; column_data[col] |= ALIGN_L; }
```

$$\left\{ \binom{i_4}{d_1} = \mathit{fold}_a\binom{i_3}{d_0}\left(\mathbf{i} = i_3 \wedge \mathbf{e} = C \Rightarrow \mathbf{c}_1{++}\right) \right\}$$

```
13:    while (i < end && a[i] == '-') { i++; dashes++; }
```

$$\left\{ \binom{i_5}{d_2} = \mathit{fold}_a\binom{i_4}{d_1}\left(\mathbf{i} < e_1 \wedge \mathbf{e} = D \Rightarrow \mathbf{c}_1{++}\right) \right\}$$

```
14:    if (a[i] == ':') { i++; dashes++; column_data[col] |= ALIGN_R; }
```

$$\left\{ \binom{i_6}{d_3} = \mathit{fold}_a\binom{i_5}{d_2}\left(\mathbf{i} = i_5 \wedge \mathbf{e} = C \Rightarrow \mathbf{c}_1{++}\right) \right\}$$

```
15:    if (i < end && a[i] != '|' && a[i] != '+') break;
16:    if (dashes < 3) break;
17:    i++;
```

$$\left\{ (i_6 \geq \mathit{end}_1 \vee a[i_6] = P \vee a[i_6] = A) \wedge d_3 \geq 3 \wedge i_7 = i_6 + 1 \wedge c_1 = c_0 + 1 \right\}$$

```
18:  }
19:  if (col < pipes) return 0;
```

$$\left\{ c_1 \geq p_2 \right\}$$

Figure 6.2: An excerpt from the Redcarpet Markdown parser with AFL annotations

semantics of the preceding code lines in the SSA form. To shorten the presentation we use the following conventions: variables `i`, `a`, `pipes`, `end`, `col`, and `dashes` are represented by (SSA-indexed) logical variables i, a, p, e, c, and d respectively; characters '`\n`', '`|`', '`:`', '`-`', and '+' by logical constants N, P, C, D, and A respectively; finally, the subscript denotes the SSA index of a variable.

The Presburger constraints such as those after line 2 are standard and we do not elaborate on them here. The first AFL-specific annotation goes after line 4: it directly reflects the loop semantics. The *fold* term encodes the computation of the number of pipes: they are computed in the counter $\mathbf{c}_1$, which gets its initial value equal to p_0, and its final value is equal to p_1. Similarly, array index $\mathbf{i}$ is initialized with i_0; and its final value is asserted to be equal to i_1. Both for counter $\mathbf{c}_1$ and for index $\mathbf{i}$ (which is a special type of a counter) their initial and final values can be both constant and symbolic: in fact, arbitrary Presburger terms are allowed.

Notice that the loop at lines 3-4 is outside of the class of loops that can be accelerated by previous approaches. In particular, the difficulty here is the combination of the iteration over arrays with the branching structure inside the loop. On the contrary, AFL can summarize the loop in a concise logical formula.

The next conditional statement at line 5, takes care of the optional pipe at the beginning of the input. The annotation shown demonstrates that conditional statements are also easily represented by *fold* terms. In particular, here the function is folded over a starting from 0; the final index is unconstrained. The branch checks that the index is 0 (to prevent going further over the array), and that the symbol at this position is '`|`'. Counter $\mathbf{c}_1$ is decremented only if these two conditions are met; otherwise, the *fold* terminates. An equivalent encoding using only array reads is possible: $(a[0] = P \ \wedge \ p_2 = p_1 - 1) \ \vee \ (a[0] \neq P \ \wedge \ p_2 = p_1)$, but this encoding involves a disjunction.

The other program statements of the motivating example are encoded in a similar fashion. The encoding shown is for one unfolding of the `for` loop at line 10; several unfoldings are encoded similarly. We have checked the resulting proof obligations with our solver for AFL formulas, called AFOLDER; it can discharge them and generate the required test input in less than 2 minutes for $n = 3$.

6.5 Theoretical Complexity

Symbolic counter machines A counter machine is a finite automaton extended by a vector $\eta = (\eta_1, \ldots, \eta_k)$ of k counters. Every counter in η stores a non-negative integer, and a counter machine can compare it to a constant, and increment/decrease its value by a constant. For the formal definition of counter machines consult, e.g. [180].

We extend counter machines to symbolic counter machines (SCMs), which accept words (arrays) of integers. We denote the symbolic value of an array cell by a special integer variable $x_{\mathbf{e}}$. Let X be a set of integer variables, where $x_{\mathbf{e}} \notin X$. An *atomic counter constraint* is a formula of the form: true, $\eta_i \approx c$ or $\eta_i \approx x$, where $c \in \mathbb{N}$, $x \in X$, and $\approx \in \{<, \leq, >, \geq, =, \neq\}$. Similarly, an *atomic input constraint* is of the form: true, $x_{\mathbf{e}} \approx c$ or $x_{\mathbf{e}} \approx x$. A *counter constraint* ν_{CC} (resp. an *input constraint* ν_{IC}) is a conjunction of atomic counter constraints (resp. atomic input constraints):

$$\begin{aligned} \nu_{\mathsf{CC}} &::= \mathsf{true} \mid \eta_i \approx c \mid \eta_i \approx x \mid \nu_{\mathsf{CC}} \wedge \nu_{\mathsf{CC}} \\ \nu_{\mathsf{IC}} &::= \mathsf{true} \mid x_{\mathbf{e}} \approx c \mid x_{\mathbf{e}} \approx x \mid \nu_{\mathsf{IC}} \wedge \nu_{\mathsf{IC}}. \end{aligned}$$

We denote by $\mathsf{CC}_k(X)$ (resp. $\mathsf{IC}(X)$) the set of all counter constraints with counters not greater than k (resp. input constraints) over variables in X.

Definition 6.1 (Symbolic counter machine)**.** *A* symbolic k-counter machine (SCM) *is a tuple* $\mathcal{C} = (\eta, X, S, \delta, \imath)$*, where:*

- $\eta = (\eta_1, \ldots, \eta_k)$ *is a vector of* k *counter variables,*
- X *is a finite set of integer variables,*
- S *is a finite set of states,*
- $\delta \subseteq S \times \mathsf{CC}_k(X) \times \mathsf{IC}(X) \times S \times \mathbb{Z}^k$ *is a transition relation,*
- $\imath \in S$ *is the initial state.*

A configuration of a k-counter SCM $\mathcal{C}$ is a tuple $\zeta = (q, \mathbf{w})$, where $q \in S$, and $\mathbf{w} \in \mathbb{Z}^k$ is a vector of non-negative integers. Let $\sigma_X : X \mapsto \mathbb{Z}$ be an interpretation of the variables in X. A SCM $\mathcal{C}$ may make a transition from a configuration $\zeta = (q, \mathbf{w})$ to a configuration $\zeta' = (q', \mathbf{w}')$

on the input $c \in \mathbb{Z}$ under the interpretation σ_X, written as $\zeta \xrightarrow{c}_{\sigma_X} \zeta'$, if there exists a transition $(q, \nu_{\mathsf{CC}}, \nu_{\mathsf{IC}}, q', \mathbf{v}) \in \delta$, such that (i) $\sigma_X \models \nu_{\mathsf{CC}}[\eta_1/\mathbf{w}(1), \cdots, \eta_k/\mathbf{w}(k)]$, (ii) $\sigma_X \models \nu_{\mathsf{IC}}[x_{\mathbf{e}}/c]$, and (iii) $\mathbf{w}' = \mathbf{w} + \mathbf{v}$. A *path* in $\mathcal{C}$ under the interpretation σ_X is a sequence of configurations $\zeta_0, \cdots, \zeta_n$, such that (a) $\zeta_0 = (\imath, \cdot)$, and (b) there exists a word $c_0, \cdots, c_{n-1} \in \mathbb{Z}^n$, such that $\forall 0 \leq i < n \,:\, \zeta_i \xrightarrow{c_i}_{\sigma_X} \zeta_{i+1}$. A SCM is called *deterministic* if δ is functional. A symbolic counter machine is *reversal-bounded* if there exists a constant $c \geq 0$ such that under all interpretations on all paths each counter makes at most c alternations between non-increasing and non-decreasing (or vice-versa).

Translation of folds to SMC The following definitions show how fold terms can be translated to SCMs. For technical convince we assume that variables and constants are the only integer terms inside folds; any AFL formula can be rewritten to this form by replacing each complex integer term T by a fresh variable x and adding the equality $x = T$ to the outermost formula.

Definition 6.2. *Let ϕ be an AFL formula, such that all integer terms in the folds of ϕ are variables or constants. We define the translation of a functional constant f of sort FSort^m, occurring in ϕ, to an SCM $\mathcal{C}(f) = (\eta, X, S, \delta, \imath)$. Let $G = \langle Q, E, \gamma \rangle$ be the edge-labeled graph for f as defined in Section 6.3.1. Then $\eta = \{\mathbf{i}, \mathbf{c}_1, \ldots, \mathbf{c}_{m-1}\}$, X are the variables in f, $S = Q$, $\imath = 0$, and for each edge $(s_1, s_2) \in E$, δ contains a transition from s_1 to s_2 labeled with a conjunction of all constraints labeling the edge. Due to the constraint on G, we have that $\mathcal{C}(f)$ is reversal-bounded.*

Thus, we can translate a *fold* term into an SCM. A *parallel composition* of SCMs captures the scenario when several *fold*s operate over the same array.

Definition 6.3. *The parallel composition (product) of two SCMs $\mathcal{C}_1$ and $\mathcal{C}_2$, where $\mathcal{C}_i = (\eta_i, X_i, S_i, \delta_i, \imath_i)$, is an SCM $\mathcal{C} = (\eta, X, S, \delta, \imath)$ such that:*

- $\eta = \eta_1\eta_2$,
- $X = X_1 \cup X_2$,
- $S = S_1 \times S_2$,
- *for each pair of transitions $(q_i, \nu_{\mathsf{CC}_i}, \nu_{\mathsf{IC}_i}, p_i, \mathbf{w}_i) \in \delta_i$, where $i = 1..2$, there is the transition $\big((q_1, q_2), \nu_{\mathsf{CC}_1} \wedge \nu_{\mathsf{CC}_2}, \nu_{\mathsf{IC}_1} \wedge \nu_{\mathsf{IC}_2}, (p_1, p_2), \mathbf{w}_1\mathbf{w}_2\big) \in \delta$, which are the only transitions in δ,*

- $\imath = (\imath_1, \imath_2)$.

Small-model property One of the fundamental questions that can be asked about a logic concerns the size of its models. The following lemma shows that models of bounded size are enough to check the satisfiability of an AFL formula.

Lemma 6.1 (Small-model property)**.** *There exists a constant $c \in \mathbb{N}$, such that an AFL formula ϕ is satisfiable iff there exists a model σ such that a) for each integer variable x in ϕ, σ maps x to an integer $\leq 2^{|\phi|^c}$, and b) for each array variable in ϕ, σ maps the variable to a sequence of $\leq 2^{|\phi|^c}$ integers, where each integer is $\leq 2^{|\phi|^c}$.*

Proof. One direction of the proof is trivial.

For the other direction, assume that ϕ has a model σ. Let X be the set of variables in ϕ. W.l.o.g. we assume that all folds are of the form

$$\begin{pmatrix} out_1 \\ \cdots \\ out_n \end{pmatrix} = fold_a \begin{pmatrix} in_1 \\ \cdots \\ in_n \end{pmatrix} F^m$$

where $out_1, \ldots, out_n, in_1, \ldots, in_n$ are integer variables.

From the model σ of ϕ we build a conjunction ψ of literals in the following way: for every atomic formula γ of the form $T = T, V^m = V^m$, or $T \leq T$ in ϕ we add a conjunct γ to ψ if σ satisfies γ, and we add a conjunct $\neg\gamma$ otherwise. Observe that σ is a model of ψ and every model of ψ is a model of ϕ. In the remaining part of the proof we show that ψ has a small model.

Let $s = |\psi|$, and note that $s \leq 3|\phi|$. Moreover, we write $\psi = \psi_f \wedge \psi_{nf}$, where ψ_f contains only literals with folds, and ψ_{nf} contains only literals without folds.

Let us assume that folds are over the same array a; we will later deal with this restriction. Let $F = \{fold_a^1, \ldots, fold_a^n\}$ be a set of folds in ψ over the array a. We translate each fold $fold_a^i \in F$ to a symbolic counter machine $\mathcal{C}_i$. Each $\mathcal{C}_i$ has at most s transitions, and the sum of counters and reversals among all $\mathcal{C}_i$ is at most s. Next, we create the product $\mathcal{C} = (\eta, S, \Sigma, \delta, \imath)$ of all the folds in F. The product counter machine $\mathcal{C}$ has at most $k = s$ counters, s^s states and transitions, and makes at most $r = s$ reversals.

In the following part of the proof, we extend the technique of [179] to show that there exist a sufficiently short path of $\mathcal{C}$. Under the interpretation σ, all variables in the counter constraints

become constants. Let $\mathbf{c} = (c_1, \dots c_n)$ be a non-decreasing vector of constants that appear in the counter constraints of $\mathcal{C}$ after fixing σ. Vector $\mathbf{c}$ gives rise to a set of regions

$$\mathcal{R} = \{[0, c_1], [c_1, c_1], [c_1 + 1, c_2 - 1], [c_2, c_2], \dots, [c_l, \infty]\}.$$

The size of $\mathcal{R}$ is at most $2\dim(\mathbf{c}) + 1 \leq 3s$. A mode of $\mathcal{C}$ is a tuple in $\mathcal{R}^k$ that describes the region of each counter. Let us observe that each counter can traverse at most $|\mathcal{R}|$ modes before it makes an additional reversal. Thus, $\mathcal{C}$ in any path can traverse at most $max = r \cdot k \cdot |\mathcal{R}| \in \mathcal{O}(s^3)$ different modes.

Let $\mathbf{w}^{in}, \mathbf{w}^{out} \in \mathbb{N}^k$ be the vectors of the initial and final values of the counters of $\mathcal{C}$; these vectors are given by the interpretation σ of the initial and output variables of folds. We know that $\mathcal{C}$ has a path $\zeta_0, \dots, \zeta_n$ from the initial configuration $\zeta_0 = (\imath, \mathbf{w}_{in})$ to the final configuration $\zeta_n = (\cdot, \mathbf{w}_{out})$ under the interpretation σ on the word $\sigma(a)$. Let $Tr = t_1, \dots, t_n \in \delta^n$ be the set of transitions corresponding to this path, i.e. $\mathcal{C}$ makes a transition from ζ_i to ζ_{i+1} by executing transition t_i.

We partition Tr into sub-sequences $Tr_1, \dots, Tr_{max}$, such that all transitions in Tr_i are fired from a configuration in mode i, and for $i < max$ the last transition in Tr_i leads to a configuration in mode $i + 1$. Let us look into some sub-path $Tr_i = t_l \dots t_{l'}$. For each transition $t \in \delta$ we mark one occurrence of t in Tr_i, provided that such a transition occurs. In this way, we mark at most $|\delta| \leq s^s$ transitions in Tr_i.

Next, we identify sub-sequences $\rho = t_m, \dots, t_n$ of Tr_i, such that (i) $m < n$, (ii) $t_m = t_n$, (iii) all transitions in ρ are unmarked, and (iv) all transitions $t_{m+1}, \dots, t_{n-1}$ are distinct. Observe that by deleting transitions $t_{m+1}, \dots, t_n$ from Tr_i we obtain a valid sequence of transition that ends in the same state, but may lead to different counter values. Let $\overline{Tr}_i$ be the sequence of transitions that results by repeatedly deleting all such sequences from Tr_i, and let S_i be the multi-set of sequences deleted in such way. Since marked configurations remain in $\overline{Tr}_i$ (and there are at most $|\delta|$ of them), and there can be at most $|\delta|$ remaining transition between any two marked configuration in $\overline{Tr}_i$, therefore $\overline{Tr}_i$ has at most $|\delta|(1 + |\delta|) = \mathcal{O}(s^{3s})$ transitions.

For each S_i we define an equivalence relation $S_i^{=}$ on the deleted transitions as follows: two deleted sequences are equivalent if they (i) have the same starting transitions, (ii) have the same end transitions, and (iii) add/subtract the same value for each counter. Note that in Tr_i a deleted sub-sequence can be substituted by an equivalent one, without changing the

final configuration, i.e. the state and the value of counters remain the same. Let u be the maximum constant which is added/subtracted in a counter update; u is given in binary, so $u \leq 2^s$. There may be at most $|\delta|$ transitions in a deleted sequence, so the total net effect on a single counter may be at most $u|\delta| = s^s 2^s$. The number of equivalence classes in $S_i^=$ is $|S_i^=| \leq |\delta|^2 (u|\delta|)^k \leq s^{2s}(s^s 2^s)^s \in \mathcal{O}(s^{2s+2s^s})$.

We construct an integer linear program LP, which expresses models of ψ that lead to some path Tr^* of $\mathcal{C}$ such that (i) Tr^* goes through the same modes as Tr, (ii) Tr^* has the same net effect on each counter, (iii) Tr^* is possibly shorter than Tr. The linear program has five parts $LP = LP_1 \wedge \ldots \wedge LP_5$.

$\mathbf{LP_1}$ In LP_1 we specify constraints for the counter values. For every counter $1 \leq j \leq k$ and mode $1 \leq i \leq max$ we create variables $w_{i,j}$ and $z_{i,j}$ which describe the value of counter j at the start and the end of mode i. Thus, $z_{max,j}$ is the value of counter j after executing Tr^*, and $w_{1,j}$ is the initial value of the counter.

For $1 \leq j \leq k, 1 \leq i \leq max$, part LP_1 contains the following equations

$$z_{i,j} = w_{i,j} + \bar{b}_{i,j} + \sum_{m=1}^{|S_i^=|} b_{i,j,m}\, y_{i,m},$$

where $\bar{b}_{i,j} \in \mathbb{Z}$ is the net effect of $\overline{Tr}_i$ on the counter j, $b_{i,j,m} \in \mathbb{Z}$ is the net effect of the sequence in the equivalence class m in $S_i^=$ on the counter j, and variable $y_{i,m}$ denotes the number of times that the sequence from class m occurs in mode i. Note that the absolute values of $\bar{b}_{i,j}$ and $b_{i,j,m}$ are at most $\mathcal{O}(2^s s^{3s})$.

Additionally, for $1 \leq i < max - 1$, we add to LP_1 the constraint:

$$w_{i+1,j} = z_{i,j} + b_{i,j}^+,$$

where $b_{i,j}^+$ is the effect of the last transition in $\overline{Tr}_i$ on the counter j. We denote by Z, W, Y the sets of the variables z, w, y, respectively.

$\mathbf{LP_2}$ In LP_2 we specify the constrain for ψ_{nf}, which is the part of ψ that does not contain folds.We create a copy X' of all the variables in X. The linear program contains a formula

$$LP_2 = \psi_{nf}[X/X'].$$

$\mathbf{LP_3}$ We link the initial and end values of counters to the variables in X'. For a counter $1 \leq j \leq k$ we add the following constraints to LP_3

$$z_{max,j} = out'_k \wedge w_{i,j} = in'_k,$$

where out_k, in_k are the variables that specify in ψ the initial and output values of counter j of $\mathcal{C}$.

LP$_4$ W.l.o.g. we assume that the counters in the folds of ψ are only compared to variables. As the result of our translation, the constants in the counter constraints of $\mathcal{C}$ arise from interpretation σ of some variables in X. Thus, we add the constraint that counter values remain within their modes. Suppose in mode i, counter j is in the region specified by the interval $[x_k, x_l]$, where $x_k, x_l \in X$. We add the following constraints to LP_4:

$$x'_l \leq w_{i,j} \leq x'_k \quad \wedge \quad x'_l \leq z_{i,j} \leq x'_k.$$

LP$_5$ Finally, we need to ensure that all the input constraints executed in Tr are satisfiable. Let $\mathcal{I}$ be the set of input constraints executed in Tr, where $|\mathcal{I}| \leq s$. For every input constraint $\nu_{\mathsf{IC}} \in \mathcal{I}$ we create an new variable v_i and add to LP_5 a linear constraint over v^i and X' that corresponds to ν_{IC}. We denote by V the set of v_i variables.

The linear program LP can be expressed in the form $\mathbf{A}\mathbf{x} \leq \mathbf{b}$, where $\mathbf{x}$ is a vector of variables in $X' \cup Z \cup W \cup Y \cup V$, and and $\mathbf{A}, \mathbf{b}$ are a matrix, and vector of integers. To determine the dimension of $\mathbf{x}$, observe that the cardinalities of X', Z, W, V are polynomial in s, while

$$|Y| \leq \sum_{i=1}^{max} |S_i^{=}| \in \mathcal{O}(s^{s^{k_0}}),$$

where k_0 is a fixed constant. Thus, $\dim(\mathbf{x}) \in \mathcal{O}(s^{s^{k_0}})$. Each of the constraints in $LP_1, \ldots, LP_2$ adds a number of constrains polynomial in s, so $\dim(\mathbf{b}) \in \mathcal{O}(s^{k_1})$, where k_1 is a fixed constant. Observe that the absolute value of the entries in $\mathbf{A}$ and $\mathbf{b}$ is at most $\mathcal{O}(2^s s^{3s})$.

We know that there exists a solution to $\mathbf{x}$ that satisfies the linear program: this solution can be obtained be assigning appropriate variables to the values of Tr and σ. By [186] we know that there also exists a solution p to $\mathbf{x}$ that assigns to every variable an element with the absolute value at most $2^{s^{k_2}}$, for some fixed constant k_2 (to satisfy the requirements of [186], we transforms the LP program to the form $\mathbf{A}'\mathbf{x}' = \mathbf{b}', \mathbf{x}' \geq 0$ by introducing new variables and constraints, whose number is at most polynomial in s. The solutions to $\mathbf{x}'$ can be mapped to the solutions in $\mathbf{x}$).

From the solution p we can construct a model σ' of ψ in the two steps. First, for integer variables in X, assign the values of X' under p. Clearly, each variable gets a value with a representation that is $\mathcal{O}(s^{k_2})$ bits.

Second, to get an assignment σ' to array a we create a computation Tr^* that has the same effect as Tr. To obtain Tr_i^*, in every reduced sub-sequence $\overline{Tr}_i$ place $p(y_{i,m})$ copies of a

sequence in the equivalence class m of $S_i^=$ after the "marked transition rule identical to the last transition rule in the sequence" (see [179]). Then $Tr^* = Tr_1^*, \ldots, Tr_{max}^*$, and the length of Tr^* is in $\mathcal{O}(2^{s^{k_3}})$, where k_3 is a fixed constant. The assignment to array a is a sequence of integers which satisfy the input constraints that path Tr^* reads. For each input constraint ψ_i read by Tr^*, we may use a solution p to v_i as a concrete value. The assignment to array a is of length $\mathcal{O}(2^{s^{k_3}})$, and each element can be represented in at most $\mathcal{O}(s^{k_2})$ bits. To complete the proof, let us observe that there can be at most $|\psi|$ arrays in the formula ψ, so having multiple arrays does not change the order of growth of σ'. □

As a consequence of Lemma 6.1 we obtain a result on the complexity of AFL satisfiability checking.

Theorem 6.1. *The satisfiability problem of* AFL *is* **PSPACE**-*complete.*

Proof. Membership. By Lemma 6.1, if an AFL formula ϕ is satisfiable, then it has a model where integer variables have value $\leq 2^{|\phi|^c}$, and arrays have length $\leq 2^{|\phi|^c}$, where each array cell stores a number $\leq 2^{|\phi|^c}$. A non-deterministic Turing machine can use a polynomial number of bits to: (i) guess the value of integer variables and store them using $|\phi|^c$ bits each, (ii) guess one-by-one the value of at most $2^{|\phi|^c}$ array cells, and simulate the *fold*s. The Turing machine needs $|\phi|^c$ bits for counting the number of simulated cells. The maximum constant used in a counter increment can be at most $2^{|\phi|}$. Then, the maximal value a *fold* counter can store after traversing the array is at most $2^{|\phi|^{c+1}}$, therefore polynomial space is also sufficient to simulate the *fold* counters.

Hardness. We reduce from the emptiness problem for the intersection of deterministic finite automata, which is **PSPACE**-complete [187]. We are given a sequence $A_1, \ldots, A_n$ of deterministic finite automata, where each automaton A_i accepts the language $\mathcal{L}(A_i)$. The problem is to decide whether $\bigcap_{i=1}^n \mathcal{L}(A_i) \neq \emptyset$. We simulate automata A_i with a *fold* expression $fold_a^i$ over a single counter, where input constraints correspond to the alphabet symbols of the automata. The expression $fold_a^i$ returns an even number on array a if and only if the interpretation of a represents a word in $\mathcal{L}(A_i)$. To check emptiness of the automata intersection, it is enough to check whether there exists an array such that all folds $fold_a^1, \ldots, fold_a^n$ return even numbers. The reduction can be done in polynomial time. □

6.5.1 Undecidable Extensions

We show that three natural extensions to our logic lead to undecidability.

Theorem 6.2. *The satifiability problem of* AFL *with an* $\exists^*\forall^*$ *quantifier prefix is undecidable.*

Proof. We prove by a reduction from Hilbert's Tenth Problem [188]; since addition is already in the logic, we only show how to encode multiplication. The following $\exists^*\forall^*$ AFL formula has a model iff array a is a repetition of z segments, and each segment is of length y and has the shape $00...01$; thus, it asserts that $x = y \cdot z$:

$$|a| = x \ \wedge \ fold_a \begin{pmatrix} 0 \\ 0 \end{pmatrix} \begin{pmatrix} \mathbf{e}{=}0 \Rightarrow skip \\ \mathbf{e}{=}1 \Rightarrow \mathbf{c}_1{++} \end{pmatrix} = \begin{pmatrix} |a| \\ z \end{pmatrix} \ \wedge$$
$$\forall j \ : \ 0 \leq j < |a| \implies fold_a \begin{pmatrix} j \\ 0 \end{pmatrix} \begin{pmatrix} \mathbf{i} \leq j{+}y \wedge \mathbf{e}{=}0 \Rightarrow skip \\ \mathbf{i} \leq j{+}y \wedge \mathbf{e}{=}1 \Rightarrow \mathbf{c}_1{++} \end{pmatrix} = \begin{pmatrix} * \\ 1 \end{pmatrix}$$

□

In [28], the following is proved about the theory of concatenation:

Theorem 6.3 ([28], Corollary 4; see also [165], Proposition 1)**.** *Solvability of equations in the theory* $\langle \{1,2\}^*, e, \circ, Lg_1, Lg_2 \rangle$*, where* $Lg_p(x) \equiv \{y \in p^* \mid y$ *has the same number of* p*'s as* $x\}$*, is undecidable.*

Corollary 6.1. *The satisfiability problem of* AFL *with the concatenation operator* $\circ$ *is undecidable.*

Proof. For an array x, we can define another array $Lg_1(x)$ in AFL as follows:

$$\begin{pmatrix} |x| \\ |Lg_1(x)| \end{pmatrix} = fold_x \begin{pmatrix} 0 \\ 0 \end{pmatrix} \begin{pmatrix} \mathbf{e}{=}1 \Rightarrow \mathbf{c}_1{++} \\ \mathbf{e}{\neq}1 \Rightarrow skip \end{pmatrix} \ \wedge \ \begin{pmatrix} |Lg_1(x)| \end{pmatrix} = fold_{Lg_1(x)} \begin{pmatrix} 0 \end{pmatrix} \begin{pmatrix} \mathbf{e}{=}1 \Rightarrow skip \end{pmatrix}$$

□

Folds in AFL formulas are deterministic i.e. guards are required to be mutually exclusive. The following lemma shows that AFL with nondeterministic folds is undecidable.

Corollary 6.2. *The satisfiability problem of* AFL *with nondeterministic folds is undecidable.*

Proof. Every nondeterministic reversal-bounded counter machine can be translated to a nondeterministic fold term, where the fold function mimics the transitions of the machine. In addition, the fold sets a designated counter to 1 when the machine accepts the input given as an array. Let

us observe that the formula where the designed counter returns 0 is satisfiable iff the counter machine is not universal. However, checking universality of a nondeterministic reversal-bounded counter machine is undecidable [189]. □

6.6 Decision Procedure

In Section 6.5 we described how a non-deterministic Turing machine can decide AFL satisfiability in **PSPACE**. Now we present a deterministic procedure that translates AFL formulas to equisatisfiable quantifier-free Presburger formulas. As a consequence of the procedure, we show that under certain restrictions satisfiability of AFL is **NP**-complete.

Deterministic procedure We are given an AFL formula ϕ such that there are at most m $fold$s over each array; clearly m can be at most $|\phi|$. We translate ϕ to a quantifier-free Presburger formula $\psi = \psi_n \wedge \psi_e \wedge \psi_l$. For the procedure we assume that there exists a fixed order $x_1 \leq \cdots \leq x_n$ on variables that appear in the counter constraints. We also assume that ϕ does not contain any array reads and writes. Array reads and writes can be rewritten by applying the procedure of [184] with linear increase in the size of the formula.

Formula ψ_n. The formula ψ_n is the part of ϕ that does not contain $fold$s.

Formula ψ_e. For an array a_j in ϕ, let $F_j = \{fold_a^1, \ldots, fold_a^m\}$ be the set of $fold$s in ϕ over a_i. We translate each $fold_a^i \in F_j$ to a symbolic counter machine $\mathcal{C}_j^i$. Each $\mathcal{C}_j^i$ has at most $|\phi|$ transitions, and the sum of the counters and the number of reversals among all $\mathcal{C}_j^i$ is at most $|\phi|$. Next, we construct the symbolic counter machine $\mathcal{C}_j$ as the product of all machines $\mathcal{C}_j^i$. The machine $\mathcal{C}_j$ has at most $k = |\phi|$ counters, $t = |\phi|^m$ transitions and makes at most $r = |\phi|$ reversals.

We translate the reachability problem of $\mathcal{C}_j$ to a quantifier-free Presburger formula ψ_e^j by applying an extension of the method described in [181]. In formula ψ_e^j, two configurations of $\mathcal{C}_j$ are described symbolically: initial ζ, and final ζ'. The formula ψ_e^j is satisfiable iff there is an array a_j such that $\mathcal{C}_j$ reaches ζ' from ζ on reading a_j. The formula ψ_e is the conjunction ψ_e^j for all arrays a_j.

The formula ψ_e^j consists of two parts $\psi_e^j = \psi_p^j \wedge \psi_c^j$. For simplicity we assume that the counter constraints of $\mathcal{C}_j$ are defined only over variables $\{x_1, \cdots, x_n\}$. By assumption, there is a fixed order $x_1 \leq \ldots \leq x_n$, which gives rise a to the set of $\leq 2|\phi| + 1$ regions

$\mathcal{R} = \{[0, x_1], [x_1, x_1], [x_1 + 1, x_2 - 1], \cdots, [c_l, \infty]\}$. As an optimization, we construct regions separately for each counter, which allows us to obtain a tighter bound on the number of regions that need to be encoded.

Each counter may traverse at most $|\mathcal{R}|$ regions before it makes a reversal, so an path of $\mathcal{C}_j$ traverses at most $max = r \cdot k \cdot |\mathcal{R}| = \mathcal{O}(|\phi|^3)$ modes. We construct an NFA $\mathcal{A}_j$ by making max copies of the control-flow structure of $\mathcal{C}_j$. Every path of $\mathcal{A}_j$ gives a correct sequence of states in $\mathcal{C}_j$, but may violate counter constraints. By using the procedure of [182] we can encode the Parikh image of $\mathcal{A}_j$ as the formula ψ_j^p that is polynomial in the size of $\mathcal{A}$. Similar to [181], the formula ψ_c^j puts additional constraints on the Parikh image to ensure that by executing the transitions of $\mathcal{A}_j$ we obtain counter values that satisfy the counter constraints of $\mathcal{C}_j$.

The size ψ_e^j is of the order $\mathcal{O}(|\phi|^3 t) = \mathcal{O}(|\phi|^{m+3})$. The formula ψ_e is the conjunction of formulas ψ_e^j for each array a_j. There can be at most $|\phi|$ arrays, so the size of ψ_e is $\mathcal{O}(|\phi|^{m+4})$.

Formula ψ_l. Finally, formula ψ_l links the initial and final configurations in ψ_e to the variables in ψ_p.

Formula size. The size of the formula ψ is $\mathcal{O}(|\phi|^{m+4})$. By keeping m constant, the encoding size is polynomial in the size of the AFL formula ϕ.

Restricted fragment of AFL We write m-AFL for formulas that have at most m *fold* expressions per array. As a consequence of the deterministic decision procedure, restriction on m reduces the complexity of deciding satisfiability.

Lemma 6.2. *The m-AFL satisfiability problem, for a fixed m, is* **NP***-complete.*

Proof. Membership follows from the decision procedure above. For hardness observe that any quantifier-free Presburger formula is an 0-AFL formula. □ □

Model generation Given a Presburger encoding ψ of an AFL formula ϕ, we may use the solution to ψ to generate a model of ϕ. The solution to ψ immediately gives us interpretation for the integer variables in ϕ. To obtain an interpretation for the array variables in ϕ, we observe that the folds are implicitly encoded in ψ as counter machines, and that the solution to ψ describes the Parikh vector for each machine. We use the method of [182] to get a concrete sequence of transitions in each counter machine that produces the specific Parikh vector. We construct a multigraph by repeating each transition in $\mathcal{A}_j$ according to its Parikh image, and then find an

Eulerian path in the multigraph. From the sequence of transitions in counter machines, and the interpretation of input constraints in ψ we obtain an interpretation for the arrays in ϕ.

6.7 Experimental Evaluation

We implemented the decision procedure described in Section 6.6 in a prototype tool AFOLDER; the tool is available at [161]. The tool is written in C++ and uses Z3 [190] as the solver for Presburger formulas. We evaluated our decision procedure on a number of testing and verification tasks described below.

The experimental results are shown in Table 6.3; all experiments were performed on a Ubuntu-14.04 64-bit machine running on an Intel Core i5-2540M CPU of 2.60GHz. For every example we report the length $len(\phi)$ of the AFL formula measured as "the number of logical operators" + "the number of branches in folds." The table also shows the number of *fold* expressions in a formula, and the maximum number of folds per array (MFPA). Next, we report the time for translating the problem to a Presburger formula, the time for solving the formula, and whether the formula is satisfiable. If this is the case, we report the length of a satisfying array generated by our tool; in case of several arrays, we show the longest.

Markdown This program is described in Section 6.4. The experiments are parametrized by the required number n of columns in the input.

perf_bench_numa This example is part of a benchmark program for non-uniform memory access (NUMA) [191]. The program maintains a list of threads, and for each thread a separate array of size 100 that describes processors assigned to the thread. The data is processed in a nested loop: the outer loop iterates over threads, and the inner loop counts the number of assigned processors. The outer loop also maintains the minimum, and maximum number of processors assigned to any thread. We model a testing scenario like in Section 6.4, where a symbolic execution tool unrolls the outer loop n times, and the inner loop is summarized by a fold expression. The testing goal is to provide a valid processor mapping such that each thread is assigned to exactly one processor. In Table 6.3 we show results for this benchmark parametrized by the number n of threads. The example scales well, since there a single fold per each processor array (see Lemma 6.2).

SV-COMP Examples "standard_minInArray" to "standard_vararg" are taken from the SV-

Table 6.3: Experimental results for AFOLDER.

Example	$len(\phi)$	folds	MFPA	transl. time	solving time	result	array length
Markdown(1)	62	6	3	$< 1s$	$< 1s$	sat	8
Markdown(2)	69	7	4	$1s$	$< 1s$	sat	14
Markdown(3)	76	8	5	$1.3s$	79s	sat	17
perf_bench_numa(10)	93	10	1	< 1s	< 1s	sat	100
perf_bench_numa(20)	183	20	1	< 1s	< 1s	sat	100
perf_bench_numa(40)	363	40	1	< 1s	< 1s	sat	100
standard_minInArray	10	3	3	< 1s	< 1s	unsat	-
linear_sea.ch_true	13	3	3	< 1s	< 1s	unsat	-
array_call3	11	2	3	< 1s	< 1s	unsat	-
standard_sentinel	14	3	3	< 1s	< 1s	unsat	-
standard_find	11	3	3	< 1s	< 1s	unsat	-
standard_vararg	11	3	3	< 1s	< 1s	unsat	-
histogram(8)	58	8	8	< 1s	1.3s	sat	9
histogram(9)	65	9	9	< 1s	6.9s	sat	10
histogram(10)	72	10	10	2s	55s	sat	11
histogram(11)	79	11	11	8s	368s	sat	12
histogram_unsat(11)	80	11	11	9s	19s	unsat	-

COMP benchmarks suite [192]. They model simple verification problems for loops, such as finding the position of an element in array, finding the minimum, or counting the number of positive elements. We model these programs as formulas that are unsatisfiable if the program is safe. Although the programs are simple, most verification tools competing in SV-COMP fail to prove their safety.

histogram We performed experiments on the histogram example in Section 6.3.3, parametrized by the number of range values. We observe that solving time grows rapidly with the number of folds. Example "histogram_unsat" is an unsatisfiable variation that requires two different counts in the same range.

7 Formal Testing: A Brief Summary

In this chapter, we shortly summarize authors's publications on formal testing. For an introduction to formal testing, the reader may refer to Section 1.4.

7.1 Compositional Specifications for ioco Testing

Model-based testing is a promising technology for black-box software and hardware testing, in which test cases are generated automatically from high-level specifications. Nowadays, systems typically consist of multiple interacting components and, due to their complexity, testing presents a considerable portion of the effort and cost in the design process. Exploiting the compositional structure of system specifications can considerably reduce the effort in model-based testing. Moreover, inferring properties about the system from testing its individual components allows the designer to reduce the amount of integration testing.

In our work [6], we study compositional properties of the **ioco**-testing theory. We propose a new approach to composition and hiding operations, inspired by contract-based design and interface theories. These operations preserve behaviors that are compatible under composition and hiding, and prune away incompatible ones. The resulting specification characterizes the input sequences for which the unit testing of components is sufficient to infer the correctness of component integration without the need for further tests. We provide a methodology that uses these results to minimize integration testing effort, but also to detect potential weaknesses in specifications. While we focus on asynchronous models and the **ioco** conformance relation, the resulting methodology can be applied to a broader class of systems.

7.2 Complete Composition Operators for ioco-Testing Theory

In [7] we extend the theory of input-output conformance with operators for merge and quotient. The former is useful when testing against multiple requirements or views. The latter can be used to generate tests for patches of an already tested system. Both operators can combine systems with different action alphabets, which is usually the case when constructing complex systems and specifications from parts, for instance different views as well as newly defined functionality of a previous version of the system.

7.3 Abstraction-driven Concolic Testing

Concolic testing is a promising method for generating test suites for large programs. However, it suffers from the path-explosion problem and often fails to find tests that cover difficult-to-reach parts of programs. In contrast, model checkers based on counterexample-guided abstraction refinement explore programs exhaustively, while failing to scale on large programs with precision. In our work [8], we present a novel method that iteratively combines concolic testing and model checking to find a test suite for a given coverage criterion. If concolic testing fails to cover some test goals, then the model checker refines its program abstraction to prove more paths infeasible, which reduces the search space for concolic testing. We have implemented our method on top of the concolic-testing tool CREST and the model checker CPACHECKER. We evaluated our tool on a collection of programs and a category of SVCOMP benchmarks. In our experiments, we observed an improvement in branch coverage compared to CREST from 48% to 63% in the best case, and from 66% to 71% on average.

8 Conclusion

The aim of this dissertation was to investigate the verification of systems by statistical and logical methods. Although these two approaches seem to be very different in nature, we demonstrated that they can be successful combined for the analysis of probabilistic systems. More precisely, we showed that logical properties (such as satisfying an LTL formula or equivalence with respect to fragments of LTL) can be inferred by sampling from a Markov chain and applying statistics. It is fascinating that such a combination is even possible, where a limited number of simulation runs is enough to prove global system properties, including properties describing infinite behavior. An inspiring, and somewhat challenging, direction for future work is to investigate combination of logical and statistical methods also for programs. The goal would be to prove that a program has a logical property by applying statistical methods. This direction of research is closely related to machine learning, and would complement the methods discussed in this dissertation.

In the following, we summarize the contribution of this thesis, and discuss future research directions.

Statistical Model Checking for Unbounded Temporal Properties To the best of our knowledge, we proposed the first statistical model-checking algorithm that can verify LTL properties by analyzing simulation runs on the fly. This is also the first application of statistical model checking to quantitative properties such as mean payoff. In the future work, we plan to extend our method to continuous-time Markov chains, as well as to Markov decision processes. Another possible direction is to increase the performance of our method on white-box models by leveraging information about the system.

Linear Distances between Markov Chains We introduced a linear-distance framework for Markov chains and considered estimating the distances in the black-box setting from simulation runs. We investigated several distances, delimiting the (in)estimability boarder for distances given topologically, logically, and by automata. As the next step, it is desirable to look for practical algorithms that would converge fast on practical benchmarks. Another direction is to characterize the largest language for which the distance can be estimated, and, dually, the smallest language that cannot be estimated.

Qualitative Analysis of Probabilistic Systems We introduced a new relation for MDPs, called combined simulation, which characterizes refinement with respect to qualitative properties. Combined simulation can be computed in quadratic time by graph algorithms. We also proposed an assume-guarantee algorithm for computing combined simulation for a composition of MDPs. Finally, we established a tight link between MDPs and two-player games, which allowed us to lift our results to two-player games. An interesting direction for future work is to consider symbolic approaches to the problem. Another possible direction is to consider other styles of compositional reasoning for concurrent systems.

Array Folds Logic We presented a new logic for arrays, called array folds logic (AFL), which extends the quantifier-free theory of arrays. The crux of our logic is the fold term, which is based on a concept well-known from functional languages. The fold term can express counting properties, which occur frequently in real-life programs. Additionally, AFL can concisely summarize loops with internal branching and counting over arrays. We analyzed the complexity of satisfiability checking for AFL formulas, and presented an efficient decision procedure via an encoding to the quantifier-free Presburger arithmetic.

In the future work, we plan to investigate possible combinations with other decidable fragments of the theory of arrays (to allow some restricted form of quantifier alternation). We also plan to automate the generation of proof obligations and the summarization of loops, and we would like to improve the efficiency of our decision procedure by implementing suitable optimization and heuristics.

Bibliography

[1] P. Daca, T. A. Henzinger, T. Petrov, and J. Kretínský, "Faster statistical model checking for unbounded temporal properties," *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, vol. 9636, pp. 112–129, 2016.

[2] P. Daca, T. A. Henzinger, J. Kretínský, and T. Petrov, "Linear distances between Markov chains," in *Intl. Conf. on Concurrency Theory (CONCUR)*, vol. 59 of *LIPIcs*, pp. 20:1–20:15, Schloss Dagstuhl, 2016.

[3] K. Chatterjee, M. Chmelik, and P. Daca, "CEGAR for qualitative analysis of probabilistic systems," in *Intl. Conf. on Computer-Aided Verification (CAV)*, 2014.

[4] K. Chatterjee, M. Chmelik, and P. Daca, "CEGAR for compositional analysis of qualitative properties in Markov decision processes," *Formal Methods in System Design (FMSD)*, vol. 47, no. 2, pp. 230–264, 2015.

[5] P. Daca, T. A. Henzinger, and A. Kupriyanov, "Array folds logic," in *Intl. Conf. on Computer-Aided Verification (CAV)*, vol. 9780 of *Lecture Notes in Computer Science*, pp. 230–248, Springer, 2016.

[6] P. Daca, T. A. Henzinger, W. Krenn, and D. Nickovic, "Compositional specifications for ioco testing," in *Intl. Conf. on Software Testing, Verification and Validation (ICST)*, 2014.

[7] N. Benes, P. Daca, T. A. Henzinger, J. Kretínský, and D. Nickovic, "Complete composition operators for ioco-testing theory," in *Intl. Symposium on Component-Based Software Engineering (CBSE)*, 2015.

[8] P. Daca, T. A. Henzinger, and A. Gupta, "Abstraction-driven concolic testing," *Intl. Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI)*, 2016.

[9] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 1999.

[10] C. Baier and J. Katoen, *Principles of model checking*. MIT Press, 2008.

[11] P. Cousot and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *Symp. on Principles of Programming Languages (POPL)*, pp. 238–252, ACM, 1977.

[12] F. Nielson, H. R. Nielson, and C. Hankin, *Principles of program analysis*. Springer, 1999.

[13] D. Kroening and O. Strichman, *Decision Procedures - An Algorithmic Point of View*. Texts in Theoretical Computer Science. An EATCS Series, Springer, 2008.

[14] R. Alur, C. Courcoubetis, T. A. Henzinger, and P. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," in *Hybrid Systems*, vol. 736 of *Lecture Notes in Computer Science*, pp. 209–229, Springer, 1992.

[15] M. Y. Vardi, "Automatic verification of probabilistic concurrent finite-state programs," in *Annual Symp. on Foundations of Computer Science (FOCS)*, pp. 327–338, 1985.

[16] R. Alur, T. A. Henzinger, and O. Kupferman, "Alternating-time temporal logic," *J. ACM*, vol. 49, no. 5, pp. 672–713, 2002.

[17] S. K. Jha, E. M. Clarke, C. J. Langmead, A. Legay, A. Platzer, and P. Zuliani, "A Bayesian approach to model checking biological systems," in *Intl. Conf. Computational Methods in Systems Biology (CMSB)*, pp. 218–234, 2009.

[18] M. Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification," in *Proceedings of the First Symposium on Logic in Computer Science*, pp. 322–331, IEEE, 1986.

[19] R. Milner, "An algebraic definition of simulation between programs," in *Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 481–489, 1971.

[20] K. G. Larsen and A. Skou, "Bisimulation through probabilistic testing," in *Symp. on Principles of Programming Languages (POPL)*, pp. 344–352, 1989.

[21] R. Alur, T. Henzinger, O. Kupferman, and M. Vardi, "Alternating refinement relations," in *Intl. Conf. on Concurrency Theory (CONCUR)*, Lecture Notes in Computer Science, pp. 163–178, Springer, 1998.

[22] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, "Symbolic model checking: 10ˆ20 states and beyond," in *Symp. on Logic in Computer Science (LICS)*, pp. 428–439, IEEE, 1990.

[23] C. A. R. Hoare, "An axiomatic basis for computer programming," *Commun. ACM*, vol. 12, no. 10, pp. 576–580, 1969.

[24] A. R. Bradley, Z. Manna, and H. B. Sipma, "What's decidable about arrays?," in *Intl. Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI)*, pp. 427–442, 2006.

[25] J. C. King, "Symbolic execution and program testing," *Commun. ACM*, vol. 19, no. 7, pp. 385–394, 1976.

[26] W. Visser, C. S. Pasareanu, and S. Khurshid, "Test input generation with java pathfinder," in *Intl. Symp. on Software Testing and Analysis (ISSTA)*, pp. 97–107, 2004.

[27] M. Davis and H. Putnam, "A computing procedure for quantification theory," *J. ACM*, vol. 7, no. 3, pp. 201–215, 1960.

[28] J. R. Büchi and S. Senger, "Definability in the existential theory of concatenation and undecidable extensions of this theory," *Math. Log. Q.*, vol. 34, no. 4, pp. 337–342, 1988.

[29] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[30] H. L. S. Younes, E. M. Clarke, and P. Zuliani, "Statistical verification of probabilistic properties with unbounded until," in *Brazilian Symp. on Formal Methods (SBMF)*, pp. 144–160, 2010.

[31] T. Brázdil, K. Chatterjee, M. Chmelík, V. Forejt, J. Křetínský, M. Z. Kwiatkowska, D. Parker, and M. Ujma, "Verification of Markov decision processes using learning

algorithms," in *Intl. Symp. Automated Technology for Verification and Analysis (ATVA)*, pp. 98–114, 2014.

[32] R. Lassaigne and S. Peyronnet, "Probabilistic verification and approximation," *Ann. Pure Appl. Logic*, vol. 152, no. 1-3, pp. 122–131, 2008.

[33] K. Sen, M. Viswanathan, and G. Agha, "On statistical model checking of stochastic systems," in *Intl. Conf. on Computer-Aided Verification (CAV)*, pp. 266–280, 2005.

[34] R. He, P. Jennings, S. Basu, A. P. Ghosh, and H. Wu, "A bounded statistical approach for model checking of unbounded until properties," in *Intl. Conf. on Automated Software Engineering (ASE)*, pp. 225–234, 2010.

[35] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *Intl. Conf. on Computer-Aided Verification (CAV)*, LNCS 1855, pp. 154–169, 2000.

[36] N. K. Patrice Godefroid and K. Sen, "Dart: directed automated random testing," in *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, Conf. on Programming Language Design and Implementation (PLDI), (New York, NY, USA), pp. 213–223, ACM, 2005.

[37] P. Habermehl, R. Iosif, and T. Vojnar, "A logic of singly indexed arrays," in *Intl. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)* (I. Cervesato, H. Veith, and A. Voronkov, eds.), vol. 5330 of *LNCS*, pp. 558–573, Springer, 2008.

[38] P. Habermehl, R. Iosif, and T. Vojnar, "What else is decidable about integer arrays?," in *Intl. Conf. on Foundations of Software Science and Computational Structures (FoSSaCS)*, pp. 474–489, 2008.

[39] F. Alberti, S. Ghilardi, and N. Sharygina, "Decision procedures for flat array properties," *J. Autom. Reasoning*, vol. 54, no. 4, pp. 327–352, 2015.

[40] B. S. Gulavani, T. A. Henzinger, Y. Kannan, A. V. Nori, and S. K. Rajamani, "Synergy: a new algorithm for property checking," in *Intl. Symp. on Foundations of Software Engineering (FSE)*, pp. 117–127, 2006.

[41] P. Godefroid, A. V. Nori, S. K. Rajamani, and S. Tetali, "Compositional may-must program analysis: unleashing the power of alternation," in *Symp. on Principles of Programming Languages (POPL)*, pp. 43–56, 2010.

[42] N. E. Beckman, A. V. Nori, S. K. Rajamani, R. J. Simmons, S. Tetali, and A. V. Thakur, "Proofs from tests," *IEEE Trans. Software Eng.*, vol. 36, no. 4, pp. 495–508, 2010.

[43] J. Tretmans, "Model based testing with labelled transition systems," in *Formal Methods and Testing*, pp. 1–38, 2008.

[44] J. Burnim and K. Sen, "Heuristics for scalable dynamic test generation," in *Intl. Conf. on Automated Software Engineering (ASE)*, pp. 443–446, 2008.

[45] P. Godefroid, M. Y. Levin, and D. A. Molnar, "Automated whitebox fuzz testing," in *Network and Distributed System Security Symp. (NDSS)*, 2008.

[46] P. Godefroid, "Compositional dynamic test generation," in *Symp. on Principles of Programming Languages (POPL)*, pp. 47–54, 2007.

[47] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre, "Lazy abstraction," in *Symp. on Principles of Programming Languages (POPL)*, pp. 58–70, ACM, 2002.

[48] A. Pnueli, "The temporal logic of programs," in *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pp. 46–57, IEEE, 1977.

[49] E. Grädel, W. Thomas, and T. Wilke, *Automata, logics, and infinite games: a guide to current research*. LNCS 2500, Springer, 2002.

[50] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Asp. Comput.*, vol. 6, no. 5, pp. 512–535, 1994.

[51] A. Aziz, V. Singhal, F. Balarin, R. Brayton, and A. Sangiovanni-Vincentelli, "It usually works: The temporal logic of stochastic systems," in *Intl. Conf. on Computer-Aided Verification (CAV)*, LNCS 939, pp. 155–165, Springer, 1995.

[52] A. Bianco and L. de Alfaro, "Model checking of probabilistic and nondeterministic systems," in *IARCS Annual Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, LNCS 1026, pp. 499–513, Springer, 1995.

[53] A. Wald, "Sequential tests of statistical hypotheses," *The Annals of Mathematical Statistics*, vol. 16, no. 2, pp. 117–186, 1945.

[54] H. L. S. Younes, "Planning and verification for stochastic processes with asynchronous events," in *American Association for Artificial Intelligence*, pp. 1001–1002, 2004.

[55] P. Bickel and K. Doksum, *Mathematical statistics: basic ideas and selected topics*. No. Bd. 1 in Mathematical Statistics: Basic Ideas and Selected Topics, Prentice Hall, 2000.

[56] H. L. S. Younes and R. G. Simmons, "Probabilistic verification of discrete event systems using acceptance sampling," in *Intl. Conf. on Computer-Aided Verification (CAV)*, pp. 223–235, Springer, 2002.

[57] K. Sen, M. Viswanathan, and G. Agha, "Statistical model checking of black-box probabilistic systems," in *Intl. Conf. on Computer-Aided Verification (CAV)*, pp. 202–215, 2004.

[58] H. L. S. Younes, M. Z. Kwiatkowska, G. Norman, and D. Parker, "Numerical vs. statistical probabilistic model checking," *Intl. Journal on Software Tools for Technology Transfer (STTT)*, vol. 8, no. 3, pp. 216–228, 2006.

[59] H. L. S. Younes and R. G. Simmons, "Statistical probabilistic model checking with a focus on time-bounded properties," *Inf. Comput.*, vol. 204, no. 9, pp. 1368–1409, 2006.

[60] C. Jégourel, A. Legay, and S. Sedwards, "A platform for high performance statistical model checking - PLASMA," in *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 498–503, 2012.

[61] P. E. Bulychev, A. David, K. G. Larsen, M. Mikucionis, D. B. Poulsen, A. Legay, and Z. Wang, "UPPAAL-SMC: statistical model checking for priced timed automata," in *Workshop on Quantitative Aspects of Programming Languages and Systems (QAPL)*, pp. 1–16, 2012.

[62] M. Z. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems," in *Intl. Conf. on Computer-Aided Verification (CAV)*, LNCS 6806, pp. 585–591, 2011.

[63] D. E. Rabih and N. Pekergin, "Statistical model checking using perfect simulation," in *Intl. Symp. Automated Technology for Verification and Analysis (ATVA)*, pp. 120–134, 2009.

[64] M. Z. Kwiatkowska, G. Norman, and D. Parker, "The PRISM benchmark suite," in *Intl. Conf. on Quantitative Evaluation of SysTems (QEST)*, pp. 203–204, 2012.

[65] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet, "Approximate probabilistic model checking," in *Intl. Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI)*, pp. 73–84, 2004.

[66] A. David, K. G. Larsen, A. Legay, M. Mikucionis, and D. B. Poulsen, "Uppaal SMC tutorial," *Intl. Journal on Software Tools for Technology Transfer (STTT)*, vol. 17, no. 4, pp. 397–415, 2015.

[67] R. Grosu and S. A. Smolka, "Monte Carlo model checking," in *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 271–286, 2005.

[68] J. Oudinet, A. Denise, M. Gaudel, R. Lassaigne, and S. Peyronnet, "Uniform Monte-Carlo model checking," in *Intl. Conf. on Fundamental Approaches to Software Engineering (FASE)*, pp. 127–140, 2011.

[69] K. Chatterjee, "Robustness of structurally equivalent concurrent parity games," in *Intl. Conf. on Foundations of Software Science and Computational Structures (FoSSaCS)*, pp. 270–285, Springer, 2012.

[70] E. Solan, "Continuity of the value of competitive Markov decision processes," *Journal of Theoretical Probability*, vol. 16, no. 4, pp. 831–845, 2003.

[71] H. Hermanns, J. Meyer-Kayser, and M. Siegle, "Multi terminal binary decision diagrams to represent and analyse continuous time Markov chains," in *3rd Int. Workshop on the Numerical Solution of Markov Chains*, pp. 188–207, Citeseer, 1999.

[72] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden, "Metrics for labeled Markov systems," in *Intl. Conf. on Concurrency Theory (CONCUR)*, pp. 258–273, 1999.

[73] N. Ferns, P. Panangaden, and D. Precup, "Metrics for finite Markov decision processes," in *American Association for Artificial Intelligence*, pp. 950–951, 2004.

[74] L. de Alfaro, R. Majumdar, V. Raman, and M. Stoelinga, "Game relations and metrics," in *Symp. on Logic in Computer Science (LICS)*, pp. 99–108, 2007.

[75] L. de Alfaro, M. Faella, and M. Stoelinga, "Linear and branching metrics for quantitative transition systems," in *Intl. Colloquium on Automata, Languages, and Programming (ICALP)*, pp. 97–109, 2004.

[76] A. Abate, "Approximation metrics based on probabilistic bisimulations for general state-space Markov processes: A survey," *Electr. Notes Theor. Comput. Sci.*, vol. 297, pp. 3–25, 2013.

[77] F. van Breugel and J. Worrell, "Approximating and computing behavioural distances in probabilistic transition systems," *Theor. Comput. Sci.*, vol. 360, no. 1-3, pp. 373–385, 2006.

[78] F. van Breugel, B. Sharma, and J. Worrell, "Approximating a behavioural pseudometric without discount for probabilistic systems," in *Intl. Conf. on Foundations of Software Science and Computational Structures (FoSSaCS)*, pp. 123–137, 2007.

[79] G. Bacci, G. Bacci, K. G. Larsen, and R. Mardare, "On-the-fly exact computation of bisimilarity distances," in *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 1–15, 2013.

[80] G. Bacci, G. Bacci, K. G. Larsen, and R. Mardare, "Computing behavioral distances, compositionally," in *Intl. Symp. on Mathematical Foundations of Computer Science (MFCS)*, pp. 74–85, 2013.

[81] G. Bacci, G. Bacci, K. G. Larsen, and R. Mardare, "The BisimDist library: Efficient computation of bisimilarity distances for Markovian models," in *Intl. Conf. on Quantitative Evaluation of SysTems (QEST)*, pp. 278–281, 2013.

[82] A. Girard and G. J. Pappas, "Approximate bisimulation: A bridge between computer science and control theory," *Eur. J. Control*, vol. 17, no. 5-6, pp. 568–578, 2011.

[83] T. Chen and S. Kiefer, "On the total variation distance of labelled Markov chains," in *Intl. Workshop Computer Science Logic (CSL)*, pp. 33:1–33:10, 2014.

[84] G. Bacci, G. Bacci, K. G. Larsen, and R. Mardare, "On the total variation distance of semi-Markov chains," in *Intl. Conf. on Foundations of Software Science and Computational Structures (FoSSaCS)*, pp. 185–199, 2015.

[85] M. Jaeger, H. Mao, K. G. Larsen, and R. Mardare, "Continuity properties of distances for Markov processes," in *Intl. Conf. on Quantitative Evaluation of SysTems (QEST)*, pp. 297–312, 2014.

[86] G. Bacci, G. Bacci, K. G. Larsen, and R. Mardare, "Converging from branching to linear metrics on Markov chains," in *ICTAC*, pp. 349–367, 2015.

[87] L. Doyen, T. A. Henzinger, and J. Raskin, "Equivalence of labeled Markov chains," *Int. J. Found. Comput. Sci.*, vol. 19, no. 3, pp. 549–563, 2008.

[88] N. Fijalkow, S. Kiefer, and M. Shirmohammadi, "Trace refinement in labelled Markov decision processes," in *Intl. Conf. on Foundations of Software Science and Computational Structures (FoSSaCS)*, pp. 303–318, 2016.

[89] S. Kiefer and A. P. Sistla, "Distinguishing hidden Markov chains," in *Symp. on Logic in Computer Science (LICS)*, 2016.

[90] P. Caspi and A. Benveniste, "Toward an approximation theory for computerised control," in *Intl. Conf. on Embedded Software (EMSOFT)*, pp. 294–304, 2002.

[91] R. Majumdar and V. S. Prabhu, "Computing the Skorokhod distance between polygonal traces," in *Intl. Conf. on Hybrid Systems: Computation and Control (HSCC)*, pp. 199–208, 2015.

[92] J. V. Deshmukh, R. Majumdar, and V. S. Prabhu, "Quantifying conformance using the Skorokhod metric," in *Intl. Conf. on Computer-Aided Verification (CAV)*, pp. 234–250, 2015.

[93] I. Tkachev and A. Abate, "On approximation metrics for linear temporal model-checking of stochastic systems," in *Intl. Conf. on Hybrid Systems: Computation and Control (HSCC)*, pp. 193–202, 2014.

[94] L. A. Goodman, "Simultaneous confidence intervals for contrasts among multinomial populations," *The Annals of Mathematical Statistics*, pp. 716–725, 1964.

[95] A. Pnueli, "In transition from global to modular temporal reasoning about programs," in *Logics and Models of Concurrent Systems*, NATO Advanced Summer Institutes F-13, pp. 123–144, Springer, 1985.

[96] C. S. Pasareanu, D. Giannakopoulou, M. G. Bobaru, J. M. Cobleigh, and H. Barringer, "Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning," *Formal Methods in System Design (FMSD)*, vol. 32, no. 3, pp. 175–205, 2008.

[97] L. de Alfaro, T. A. Henzinger, and R. Jhala, "Compositional methods for probabilistic systems," in *Intl. Conf. on Concurrency Theory (CONCUR)*, LNCS 2154, pp. 351–365, Springer, 2001.

[98] M. Z. Kwiatkowska, G. Norman, D. Parker, and H. Qu, "Assume-guarantee verification for probabilistic systems," in *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS 6015, pp. 23–37, Springer, 2010.

[99] L. Feng, M. Z. Kwiatkowska, and D. Parker, "Automated learning of probabilistic assumptions for compositional reasoning," in *Intl. Conf. on Fundamental Approaches to Software Engineering (FASE)*, LNCS 6603, pp. 2–17, Springer, 2011.

[100] S. Chaki, E. M. Clarke, N. Sinha, and P. Thati, "Automated assume-guarantee reasoning for simulation conformance," in *Intl. Conf. on Computer-Aided Verification (CAV)*, LNCS 3576, pp. 534–547, Springer, 2005.

[101] T. A. Henzinger, R. Jhala, R. Majumdar, and S. Qadeer, "Thread-modular abstraction refinement," in *Intl. Conf. on Computer-Aided Verification (CAV)*, LNCS 2725, pp. 262–274, Springer, 2003.

[102] K. Chatterjee, L. de Alfaro, M. Faella, R. Majumdar, and V. Raman, "Code-aware resource management," *Formal Methods in System Design (FMSD)*, vol. 42, no. 2, pp. 146–174, 2013.

[103] A. Pogosyants, R. Segala, and N. Lynch, "Verification of the randomized consensus algorithm of Aspnes and Herlihy: a case study," *Distributed Computing*, vol. 13, no. 3, pp. 155–186, 2000.

[104] M. Stoelinga, "Fun with FireWire: Experiments with verifying the IEEE1394 root contention protocol," in *Formal Aspects of Computing*, 2002.

[105] T. A. Henzinger, R. Jhala, and R. Majumdar, "Counterexample-guided control," in *Intl. Colloquium on Automata, Languages, and Programming (ICALP)*, LNCS 2719, pp. 886–902, Springer, 2003.

[106] R. Chadha and M. Viswanathan, "A counterexample-guided abstraction-refinement framework for Markov decision processes," *Trans. on Computational Logic (TOCL)*, p. 1, 2010.

[107] H. Hermanns, B. Wachter, and L. Zhang, "Probabilistic CEGAR," in *Intl. Conf. on Computer-Aided Verification (CAV)*, LNCS 5123, pp. 162–175, Springer, 2008.

[108] M. Z. Kwiatkowska, G. Norman, and D. Parker, "Game-based abstraction for Markov decision processes," in *Intl. Conf. on Quantitative Evaluation of SysTems (QEST)*, pp. 157–166, 2006.

[109] K. Etessami, M. Z. Kwiatkowska, M. Y. Vardi, and M. Yannakakis, "Multi-objective model checking of Markov decision processes," *Logical Methods in Computer Science*, vol. 4, no. 4, 2008.

[110] A. Komuravelli, C. S. Pasareanu, and E. M. Clarke, "Assume-guarantee abstraction refinement for probabilistic systems," in *Intl. Conf. on Computer-Aided Verification (CAV)*, LNCS 7358, pp. 310–326, Springer, 2012.

[111] P. R. D'Argenio, B. Jeannet, H. E. Jensen, and K. G. Larsen, "Reachability analysis of probabilistic systems by successive refinements," in *Process Algebra and Probabilistic Methods, Performance Modeling and Verification: Joint International Workshop, PAPM-PROBMIV*, LNCS 2165, pp. 39–56, Springer, 2001.

[112] P. R. D'Argenio, B. Jeannet, H. E. Jensen, and K. G. Larsen, "Reduction and refinement strategies for probabilistic analysis," in *Process Algebra and Probabilistic Methods, Performance Modeling and Verification: Joint International Workshop, PAPM-PROBMIV*, LNCS 2399, pp. 57–76, Springer, 2002.

[113] B. Jeannet, P. d'Argenio, and K. Larsen, "Rapture: A tool for verifying Markov decision processes," *Tools Day*, vol. 2, p. 149, 2002.

[114] R. Cleaveland and B. Steffen, "Computing behavioural relations, logically," in *Intl. Colloquium on Automata, Languages, and Programming (ICALP)*, LNCS 510, pp. 127–138, Springer, 1991.

[115] C. Courcoubetis and M. Yannakakis, "The complexity of probabilistic verification," *J. ACM*, vol. 42, no. 4, pp. 857–907, 1995.

[116] K. Chatterjee and M. Henzinger, "Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification," in *Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pp. 1318–1336, 2011.

[117] K. Chatterjee and M. Henzinger, "An $O(n^2)$ time algorithm for alternating Büchi games," in *Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pp. 1386–1399, 2012.

[118] K. Chatterjee, M. Henzinger, M. Joglekar, and N. Shah, "Symbolic algorithms for qualitative analysis of Markov decision processes with Büchi objectives," *Formal Methods in System Design (FMSD)*, vol. 42, no. 3, pp. 301–327, 2013.

[119] K. Chatterjee and M. Henzinger, "Efficient and dynamic algorithms for alternating Büchi games and maximal end-component decomposition," *Journal of the ACM (JACM)*, 2014.

[120] K. Chatterjee and J. Lacki, "Faster algorithms for Markov decision processes with low treewidth," in *Intl. Conf. on Computer-Aided Verification (CAV)*, pp. 543–558, 2013.

[121] K. Chatterjee, M. Jurdziński, and T. A. Henzinger, "Simple stochastic parity games," in *Intl. Workshop Computer Science Logic (CSL)*, vol. 2803 of *LNCS*, pp. 100–113, Springer, 2003.

[122] K. Chatterjee, M. Jurdziński, and T. A. Henzinger, "Quantitative stochastic parity games," in *Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pp. 121–130, SIAM, 2004.

[123] K. Chatterjee, L. de Alfaro, and T. A. Henzinger, "The complexity of stochastic Rabin and Streett games," in *Intl. Colloquium on Automata, Languages, and Programming (ICALP)*, pp. 878–890, LNCS 3580, Springer, 2005.

[124] K. Chatterjee, *Stochastic ω-Regular Games*. PhD thesis, UC Berkeley, 2007.

[125] K. Chatterjee, "The complexity of stochastic müller games," *Inf. Comput.*, vol. 211, pp. 29–48, 2012.

[126] L. de Alfaro, T. A. Henzinger, and O. Kupferman, "Concurrent reachability games," in *Annual Symp. on Foundations of Computer Science (FOCS)*, pp. 564–575, 1998.

[127] K. Chatterjee, L. de Alfaro, and T. A. Henzinger, "Qualitative concurrent parity games," *Trans. on Computational Logic (TOCL)*, vol. 12, no. 4, p. 28, 2011.

[128] K. Chatterjee, L. de Alfaro, and T. A. Henzinger, "Strategy improvement in concurrent reachability games," in *Intl. Conf. on Quantitative Evaluation of SysTems (QEST)*, pp. 291–300, IEEE, 2006.

[129] K. Chatterjee, L. de Alfaro, and T. A. Henzinger, "The complexity of quantitative concurrent parity games," in *Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pp. 678–687, ACM-SIAM, 2006.

[130] K. Chatterjee, "Qualitative concurrent parity games: Bounded rationality," in *Intl. Conf. on Concurrency Theory (CONCUR)*, pp. 544–559, 2014.

[131] K. Chatterjee and R. Ibsen-Jensen, "The value 1 problem under finite-memory strategies for concurrent mean-payoff games," in *Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pp. 1018–1029, 2015.

[132] K. Chatterjee and R. Ibsen-Jensen, "Qualitative analysis of concurrent mean-payoff games," *Inf. Comput.*, 2015.

[133] C. Baier, N. Bertrand, and M. Größer, "On decision problems for probabilistic Büchi automata," in *Intl. Conf. on Foundations of Software Science and Computational Structures (FoSSaCS)*, LNCS 4962, pp. 287–301, Springer, 2008.

[134] K. Chatterjee and M. Tracol, "Decidable problems for probabilistic automata on infinite words," in *Symp. on Logic in Computer Science (LICS)*, pp. 185–194, 2012.

[135] K. Chatterjee, M. Chmelik, and M. Tracol, "What is decidable about partially observable Markov decision processes with omega-regular objectives," in *Intl. Workshop Computer Science Logic (CSL)*, 2013.

[136] K. Chatterjee, L. Doyen, and T. A. Henzinger, "Qualitative analysis of partially-observable Markov decision processes," in *Intl. Symp. on Mathematical Foundations of Computer Science (MFCS)*, LNCS 6281, pp. 258–269, Springer, 2010.

[137] K. Chatterjee and M. Chmelik, "Pomdps under probabilistic semantics," *Artif. Intell.*, vol. 221, pp. 46–72, 2015.

[138] K. Chatterjee, L. Doyen, T. A. Henzinger, and J. Raskin, "Algorithms for omega-regular games with imperfect information," in *Intl. Workshop Computer Science Logic (CSL)*, pp. 287–302, LNCS 4207, Springer, 2006.

[139] N. Bertrand, B. Genest, and H. Gimbert, "Qualitative determinacy and decidability of stochastic games with signals," in *Symp. on Logic in Computer Science (LICS)*, pp. 319–328, IEEE, 2009.

[140] K. Chatterjee and L. Doyen, "Partial-observation stochastic games: How to win when belief fails," in *Symp. on Logic in Computer Science (LICS)*, pp. 175–184, IEEE, 2012.

[141] K. Chatterjee, L. Doyen, and T. A. Henzinger, "A survey of partial-observation stochastic parity games," *Formal Methods in System Design (FMSD)*, vol. 43, no. 2, pp. 268–284, 2013.

[142] S. Nain and M. Y. Vardi, "Solving partial-information stochastic parity games," in *Symp. on Logic in Computer Science (LICS)*, pp. 341–348, 2013.

[143] K. Chatterjee, L. Doyen, S. Nain, and M. Y. Vardi, "The complexity of partial-observation stochastic parity games with finite-memory strategies," in *Intl. Conf. on Foundations of Software Science and Computational Structures (FoSSaCS)*, pp. 242–257, 2014.

[144] K. Chatterjee and L. Doyen, "Games with a weak adversary," in *Intl. Colloquium on Automata, Languages, and Programming (ICALP)*, pp. 110–121, 2014.

[145] P. Bouyer, T. Brihaye, M. Jurdzinski, and Q. Menet, "Almost-sure model-checking of reactive timed automata," in *Intl. Conf. on Quantitative Evaluation of SysTems (QEST)*, pp. 138–147, 2012.

[146] C. Baier, N. Bertrand, P. Bouyer, T. Brihaye, and M. Größer, "Almost-sure model checking of infinite paths in one-clock timed automata," in *Symp. on Logic in Computer Science (LICS)*, pp. 217–226, 2008.

[147] R. Alur and T. A. Henzinger, "Computer-aided verification," 2004. Unpublished.

[148] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke, "Computing simulations on finite and infinite graphs," in *Annual Symp. on Foundations of Computer Science (FOCS)*, pp. 453–462, 1995.

[149] K. Chatterjee, S. Chaubal, and P. Kamath, "Faster algorithms for alternating refinement relations," in *Intl. Workshop Computer Science Logic (CSL)*, LIPIcs 16, pp. 167–182, Schloss Dagstuhl, 2012.

[150] N. Immerman, "Number of quantifiers is better than number of tape cells," *Journal of Computer and System Sciences*, vol. 22, pp. 384–406, 1981.

[151] C. Beeri, "On the membership problem for functional and multivalued dependencies in relational databases," *ACM Trans. on Database Systems*, vol. 5, pp. 241–259, 1980.

[152] K. Chatterjee, L. Doyen, H. Gimbert, and T. A. Henzinger, "Randomness for free," in *Intl. Symp. on Mathematical Foundations of Computer Science (MFCS)*, pp. 246–257, 2010.

[153] K. Chatterjee, L. de Alfaro, M. Faella, and A. Legay, "Qualitative logics and equivalences for probabilistic systems," *Logical Methods in Computer Science*, vol. 5, no. 2, 2009.

[154] S. Schewe, "Tighter bounds for the determinisation of büchi automata," in *Intl. Conf. on Foundations of Software Science and Computational Structures (FoSSaCS)*, pp. 167–181, 2009.

[155] W. Zielonka, "Infinite games on finitely coloured graphs with applications to automata on infinite trees," in *Theoretical Computer Science*, vol. 200(1-2), pp. 135–183, 1998.

[156] A. Itai and M. Rodeh, "Symmetry breaking in distributed networks," *Information and Computation*, vol. 88, no. 1, 1990.

[157] G. L. Peterson, "Myths about the mutual exclusion problem," *Information Processing Letters*, vol. 12, no. 3, pp. 115–116, 1981.

[158] R. Segala and N. A. Lynch, "Probabilistic simulations for probabilistic processes," *Nord. J. Comput.*, vol. 2, no. 2, pp. 250–273, 1995.

[159] B. K. Szymanski, "A simple solution to Lamport's concurrent programming problem with linear wait," in *International conference on Supercomputing (ICS)*, pp. 621–626, 1988.

[160] P. Cerný, M. Chmelik, T. A. Henzinger, and A. Radhakrishna, "Interface simulation distances," in *Proceedings Fourth International Symposium on Games, Automata, Logics and Formal Verification (GandALF)*, EPTCS 96, pp. 29–42, 2012.

[161] P. Daca, "AFolder." https://github.com/pdaca/AFolder.

[162] M. Zhou, F. He, B.-Y. Wang, M. Gu, and J. Sun, "Array theory of bounded elements and its applications," *Journal of automated reasoning*, vol. 52, no. 4, pp. 379–405, 2014.

[163] S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli, "Decision procedures for extensions of the theory of arrays," *Ann. Math. Artif. Intell.*, vol. 50, no. 3-4, pp. 231–254, 2007.

[164] L. M. de Moura and N. Bjørner, "Generalized, efficient array decision procedures," in *Formal Methods in Computer Aided Design (FMCAD)*, pp. 45–52, 2009.

[165] C. A. Furia, "What's decidable about sequences?," in *Intl. Symp. Automated Technology for Verification and Analysis (ATVA)*, pp. 128–142, 2010.

[166] A. W. Lin and P. Barceló, "String solving with word equations and transducers: towards a logic for analysing mutation XSS," in *Symp. on Principles of Programming Languages (POPL)*, pp. 123–136, 2016.

[167] G. S. Makanin, "The problem of solvability of equations in a free semigroup," *Sbornik: Mathematics*, vol. 32, no. 2, pp. 129–198, 1977.

[168] H. Comon and Y. Jurski, "Multiple counters automata, safety analysis and presburger arithmetic," in *Intl. Conf. on Computer-Aided Verification (CAV)*, pp. 268–279, 1998.

[169] M. Bozga, R. Iosif, and F. Konecný, "Fast acceleration of ultimately periodic relations," in *Intl. Conf. on Computer-Aided Verification (CAV)*, pp. 227–242, 2010.

[170] J. Knoop, L. Kovács, and J. Zwirchmayr, "Symbolic loop bound computation for WCET analysis," in *Perspectives of Systems Informatics (PSI)*, pp. 227–242, 2011.

[171] P. Saxena, P. Poosankam, S. McCamant, and D. Song, "Loop-extended symbolic execution on binary programs," in *Intl. Symp. on Software Testing and Analysis (ISSTA)*, pp. 225–236, 2009.

[172] M. Bozga, R. Iosif, F. Konecný, and T. Vojnar, "Tool demonstration of the FLATA counter automata toolset," in *Intl. Workshop on Invariant Generation (WING)*, p. 75, 2010.

[173] P. Godefroid and D. Luchaup, "Automatic partial loop summarization in dynamic test generation," in *Intl. Symp. on Software Testing and Analysis (ISSTA)*, pp. 23–33, 2011.

[174] H. Hojjat, R. Iosif, F. Konecný, V. Kuncak, and P. Rümmer, "Accelerating interpolants," in *Intl. Symp. Automated Technology for Verification and Analysis (ATVA)*, pp. 187–202, 2012.

[175] M. Bozga, P. Habermehl, R. Iosif, F. Konecný, and T. Vojnar, "Automatic verification of integer array programs," in *Intl. Conf. on Computer-Aided Verification (CAV)*, pp. 157–172, 2009.

[176] F. Alberti, S. Ghilardi, and N. Sharygina, "Booster: An acceleration-based verification framework for array programs," in *Intl. Symp. Automated Technology for Verification and Analysis (ATVA)*, pp. 18–23, 2014.

[177] F. Alberti, S. Ghilardi, and N. Sharygina, "A new acceleration-based combination framework for array properties," in *Intl. Symp. on Frontiers of Combining Systems*, pp. 169–185, 2015.

[178] O. H. Ibarra, "Reversal-bounded multicounter machines and their decision problems," *J. ACM*, vol. 25, no. 1, pp. 116–133, 1978.

[179] E. M. Gurari and O. H. Ibarra, "The complexity of decision problems for finite-turn multicounter machines," in *Automata, Languages and Programming*, pp. 495–505, 1981.

[180] O. H. Ibarra, J. Su, Z. Dang, T. Bultan, and R. A. Kemmerer, "Counter machines and verification problems," *Theor. Comput. Sci.*, vol. 289, no. 1, pp. 165–189, 2002.

[181] M. Hague and A. W. Lin, "Model checking recursive programs with numeric data types," in *Intl. Conf. on Computer-Aided Verification (CAV)*, pp. 743–759, 2011.

[182] H. Seidl, T. Schwentick, A. Muscholl, and P. Habermehl, "Counting in trees for free," in *Automata, Languages and Programming*, pp. 1136–1149, Springer, 2004.

[183] J. McCarthy, "Towards a mathematical science of computation," in *Intl. Federation for Information Processing Congress*, pp. 21–28, 1962.

[184] N. Suzuki and D. Jefferson, "Verification decidability of presburger array programs," *J. ACM*, vol. 27, no. 1, pp. 191–205, 1980.

[185] "Markdown parser of the Redcarpet project." https://github.com/vmg/redcarpet/blob/master/ext/redcarpet/markdown.c. Accessed: 2016-01-20.

[186] C. H. Papadimitriou, "On the complexity of integer programming," *J. ACM*, vol. 28, no. 4, pp. 765–768, 1981.

[187] D. Kozen, "Lower bounds for natural proof systems," in *Annual Symp. on Foundations of Computer Science (FOCS)*, pp. 254–266, 1977.

[188] J. V. Matijasevič, "The Diophantineness of enumerable sets," *Dokl. Akad. Nauk SSSR*, vol. 191, pp. 279–282, 1970.

[189] B. S. Baker and R. V. Book, "Reversal-bounded multipushdown machines," *J. Comput. Syst. Sci.*, vol. 8, no. 3, pp. 315–332, 1974.

[190] L. M. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2008.

[191] "Benchmark for non-uniform memory access (NUMA)." http://lxr.free-electrons.com/source/tools/perf/bench/numa.c?v=3.9. Accessed: 2016-01-29.

[192] D. Beyer, "Software verification and verifiable witnesses - (report on SV-COMP 2015)," in *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 401–416, 2015.